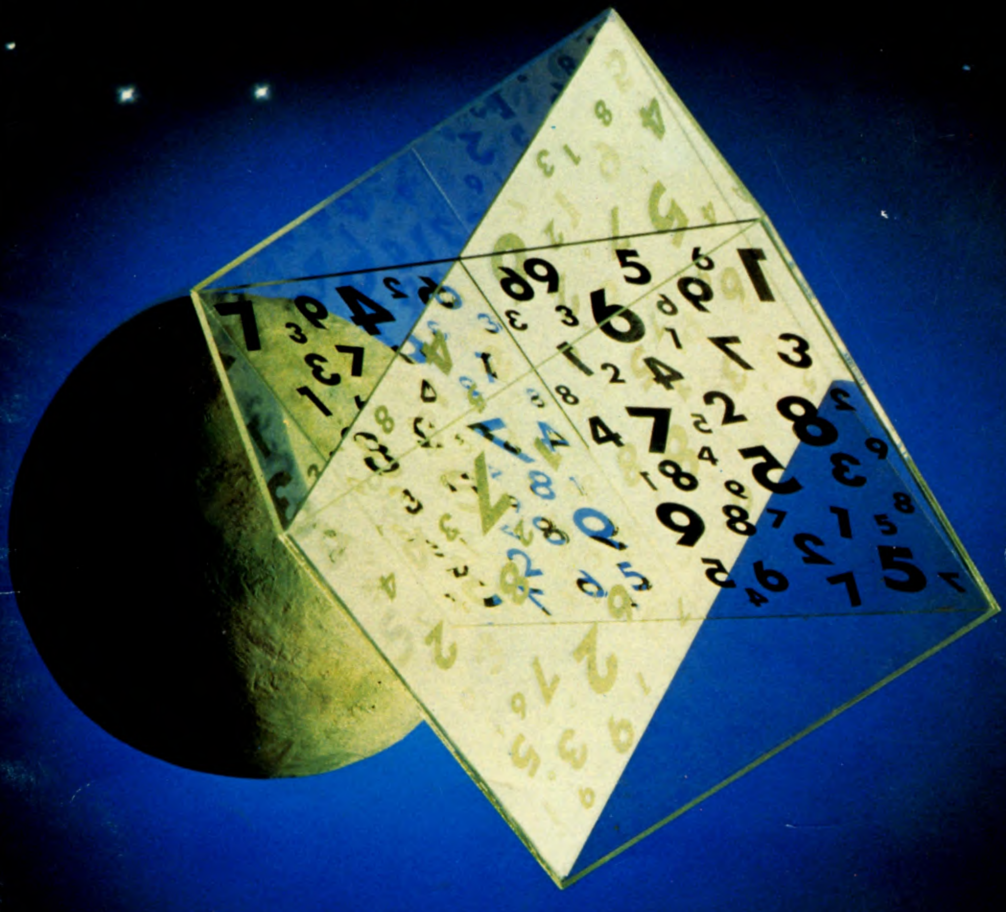
A MARSHALL CAVENDISH 5 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE



UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95

INPUT

Vol 1 No 5

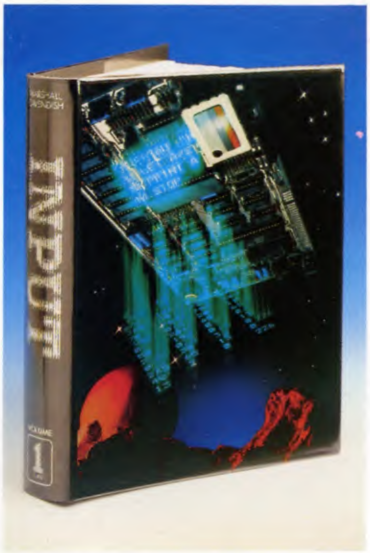
INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front Cover, Dave King. Pages 129, 130, 134, Peter Bentley. Pages 136, 138, 140, Graeme Harris. Page 142, Graeme Harris, David Lloyd. Pages 144, 146, 148, 150, Ian Craig. Pages 152, 154, Dick Ward. Page 156, Dave King. Pages 158, 160, Dave King.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.45) for each binder to the address below:

Marshall Cavendish Services Ltd,
Department 980, Newtown Road,
Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Gordon and Gotch Ltd, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Copies of any part of INPUT can be obtained from the following addresses at the regular cover price, with no extra charge for postage and packing:

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services,
Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent

COPIES BY POST

Our Subscription Department can supply your copies direct to you regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. These rates apply anywhere in the world. Send your order, with payment to:

Subscription Department, Marshall Cavendish Services Ltd,
Newtown Road, Hove, Sussex BN3 7DN

Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

S SPECTRUM 16K, 48K, 128, and +

C COMMODORE 64 and 128

ACORN ELECTRON, BBC B and B+

DRAGON 32 and 64

S ZX81

C VIC 20

T TANDY TRS80 COLOUR COMPUTER

WHAT DO I DO NEXT?

- USING INPUTS AND PROMPTS
- QUICKER METHODS WITH INKEY$ OR GET$
- SCREEN DRAWING PROGRAM
- PASSWORD PROGRAM

Before you can get a computer to respond to your wishes, you'll have to tell it what those wishes are. The way to do this is fundamental to many types of program

With very few exceptions, computers do not just tick along by themselves, giving out information. Nearly all programs, from games to business and scientific applications, require some sort of input from the user. But there are different types of information which can be given to the computer, and there are a number of ways in which this can be done in different•types of program.

The information may be as simple as pressing a cursor key to direct a missile base. Or it may be keying in numbers or text—say for a quiz, database, accounts program or scientific calculation. In the case of a game the main necessity is that the computer reacts fast. But in other applications, the ability to change what you have keyed in, before committing it to the computer's memory, is more important.

SIMPLE INPUTS

One of the first programs that everyone writes, which illustrates the use of INPUT, is along the lines of:

```
1Ø PRINT "WHAT IS YOUR NAME?"
2Ø INPUT N$
3Ø PRINT "HELLO□";N$
```

When the computer comes across the INPUT statement it waits for you to key something in. In most cases, everything which is typed in, up to the next RETURN or ENTER, is placed in the variable—N$ in this case.

In this example, the variable is a string. But numeric variables can be used as well. The following program shows how a list of five names and ages is built up using simple INPUT commands. (This program uses two arrays to store information, so look at pages 152 to 155 if you're not familiar with how these work.)

```
5 DIM NAME$(5), AGE(5)
1Ø FOR N=1 TO 5
2Ø PRINT "NAME:"
3Ø INPUT NAME$(N)
4Ø PRINT "AGE:"
5Ø INPUT AGE(N)
6Ø NEXT
7Ø FOR N=1 TO 5
8Ø PRINT NAME$(N);"□IS□";AGE(N);
   "□YEARS OLD"
9Ø NEXT
```

```
5 DIM NAME$(5), AGE(5)
1Ø FOR N=1 TO 5
2Ø PRINT "NAME:"
3Ø INPUT NAME$(N)
4Ø PRINT "AGE:"
5Ø INPUT AGE(N)
6Ø NEXT N
```

```
7Ø FOR N=1 TO 5
8Ø PRINT NAME$(N);"□IS";AGE(N);
   "YEARS OLD"
9Ø NEXT N
```

```
5 DIM n$ (5,1Ø): DIM a (5)
1Ø FOR n=1 TO 5
2Ø PRINT "Name:□"
3Ø INPUT n$ (n)
4Ø PRINT "Age:□"
5Ø INPUT a (n)
55 CLS:NEXT n
6Ø FOR n=1 TO 5
7Ø PRINT n$ (n); "□is□"; a(n);
   "□years old"
8Ø NEXT n
```

On the ZX81, type entirely in capitals. Omit Lines 5 and 55, and add:

```
5 DIM N$ (5,1Ø)
7 DIM A (5)
55 CLS
57 NEXT N
```

It is important that the right type of information is given. If you tried to type in a name when asked for an age, the program stops and PRINTs an error message (except on the Acorn). Running the program again may mean that you lose everything already entered, and have to start from scratch. That is not too much of a problem with only five entries, but in a large program it can be annoying.

USING PROMPTS

The program above may seem pretty simple, but even that little program contains one important feature. That is the *prompt* contained in the first two PRINT statements. A prompt is a message telling you that the computer wants you to enter something.

If you delete Lines 2Ø and 4Ø, the program still RUNs. With most computers you get some sort of prompt on the screen anyway. For example, many computers display a question mark.

If you are familiar with the computer, the INPUT prompt is sometimes enough to remind you that something needs to be entered. But it is not enough to tell you what type of information. Imagine you are a stranger to the program, perhaps with no experience of the computer being used and no working knowledge of the program. A question mark is just baffling.

Good use of prompts is of benefit even with your own programs, as it is very easy to forget exactly how the information should be entered. Entering dates is a good example. Most accounting programs, and many others, require dates to be entered. In many cases this information is used by the program, either in search routines when you want to find a specific entry, or to sort the entries into chronological order. If you do this, you must ensure that you always enter dates in a standard form.

A common method is to use six digit dates, sometimes with obliques or fullstops to separate the day, month and year. So a typical INPUT routine may look something like this:

```
1ØØ PRINT "ENTER DATE (IN FORMAT
   DD/MM/YY)"
11Ø INPUT D$
```

This reminds you, not only that a date is needed, but also in what form it must be entered. If you wanted to enter the 3rd September 1945, you would actually type: Ø3/Ø9/45. You can choose any format you like as long as you tell the user what you want.

The use of the INPUT command in the programs shown so far is pretty simple. But it can be made even simpler, because with many computers you can incorporate the prompt in the INPUT command itself. (On the Commodores the length of the prompt is limited to 2Ø characters.) Going back to the name and age program, you can delete Lines 2Ø and 4Ø, and rewrite 3Ø and 5Ø as:

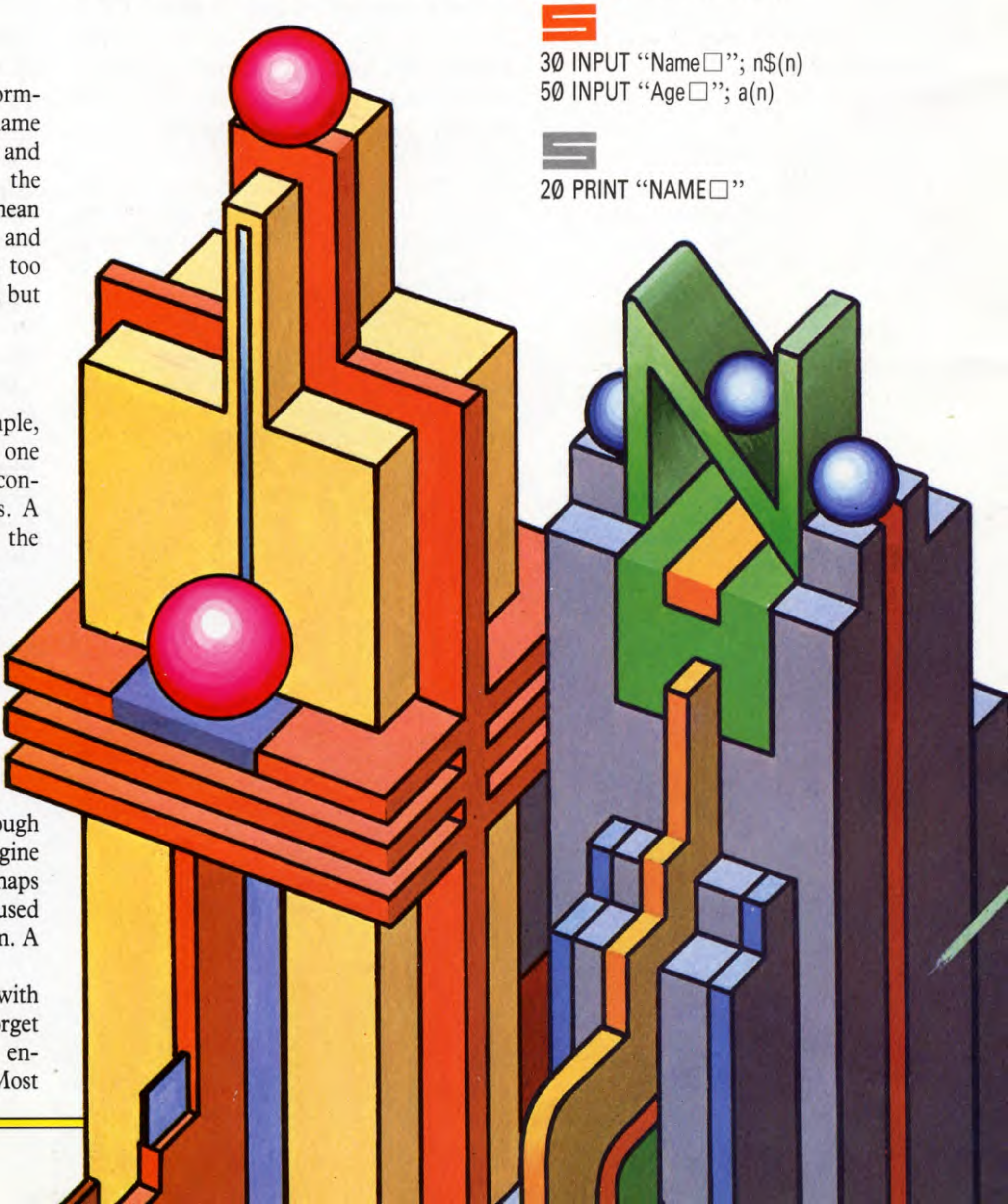
```
3Ø INPUT "NAME□"NAME$(N)
5Ø INPUT "AGE□"AGE(N)
```

```
3Ø INPUT "NAME□"; NAME$(N)
5Ø INPUT "AGE□"; AGE(N)
```

```
3Ø INPUT "NAME"; NAME$(N)
5Ø INPUT "AGE"; AGE(N)
```

```
3Ø INPUT "Name□"; n$(n)
5Ø INPUT "Age□"; a(n)
```

```
2Ø PRINT "NAME□"
```



```
3Ø INPUT N$ (N)
4Ø PRINT "AGE□"
5Ø INPUT A(N)
```

Note the spaces inside the quote marks. With many computers, your keyboard entry is PRINTed straight after the prompt, so a space helps make things clearer. However, the Dragon and Tandy PRINT the spaces for you.

This is not the only way the routine can be condensed. Normally, more than one item can be entered, using just a single INPUT command—the only computer on which this is not possible is the ZX81. The variables into which the information is fed are separated by commas. So you could replace Lines 3Ø and 5Ø (remember to delete Line 5Ø) above with this single line:

```
3Ø INPUT "ENTER NAME AND AGE□"
   NAME$(N),AGE(N)
```

```
3Ø INPUT "ENTER NAME AND AGE□";
   NAME$(N),AGE(N)
```

```
3Ø INPUT "ENTER NAME AND AGE";
   NAME$(N),AGE(N)
```

```
3Ø INPUT "Enter name and age□";
   n$(n),a(n)
```

How you actually enter the two pieces of information is different on each computer. On the Spectrum press ENTER after each INPUT. On the Dragon, Tandy, Acorn and Commodores you can either use the Spectrum method or else type in the name and age separated by a comma and only then press ENTER.

When you use a single prompt, it is important that it tells you everything you need to know about what entries are required.

Alternatively, on the Acorn and Spectrum, you can always split the prompt again. As long as each prompt is kept in quotation marks, the computer just PRINTs them, and does not treat them as variables. So, Line 3Ø can be rewritten again:

```
3Ø INPUT "NAME□" NAME$(N), "AGE□"
   AGE(N)
```

```
3Ø INPUT "Name"; n$ (n); "age";
   a (n)
```

However, note that it is not possible to do this on the Commodores, Dragon, Tandy or ZX81.

To make the display a little tidier, INPUT prompts can usually be positioned using the normal TAB commands (see pages 117 to 123).

INPUTTING A WHOLE LINE

The main problem you are likely to come across with INPUT is when you are entering strings containing commas or with spaces at the beginning. If you want a single variable to contain an address, for example, you need to include commas. And spaces at the beginning might be handy for neat screen presentation.

The problem is that most computers ignore the leading spaces, although they have no trouble with those between words. And frequently, they also ignore anything occurring after a comma, so you could find large portions of your information going missing.

Fortunately, some computers have a command which solves this problem, and the Spectrum has a way of avoiding it.

The most common solution is to include the entry in quotation marks. The Spectrum, for instance, automatically PRINTs quotation marks when a string variable is used in an INPUT command. On the other computers they have to be typed in by the user, so any prompt should point out when they are necessary.

Another solution, available on the Acorn, Dragon and Tandy computers, is to use the INPUT LINE or LINE INPUT command. This is used in exactly the same way as INPUT:

```
1Ø INPUT LINE "Please enter your
   address□", A$
```

```
1Ø LINE INPUT "PLEASE ENTER YOUR
   ADDRESS□"; A$
```

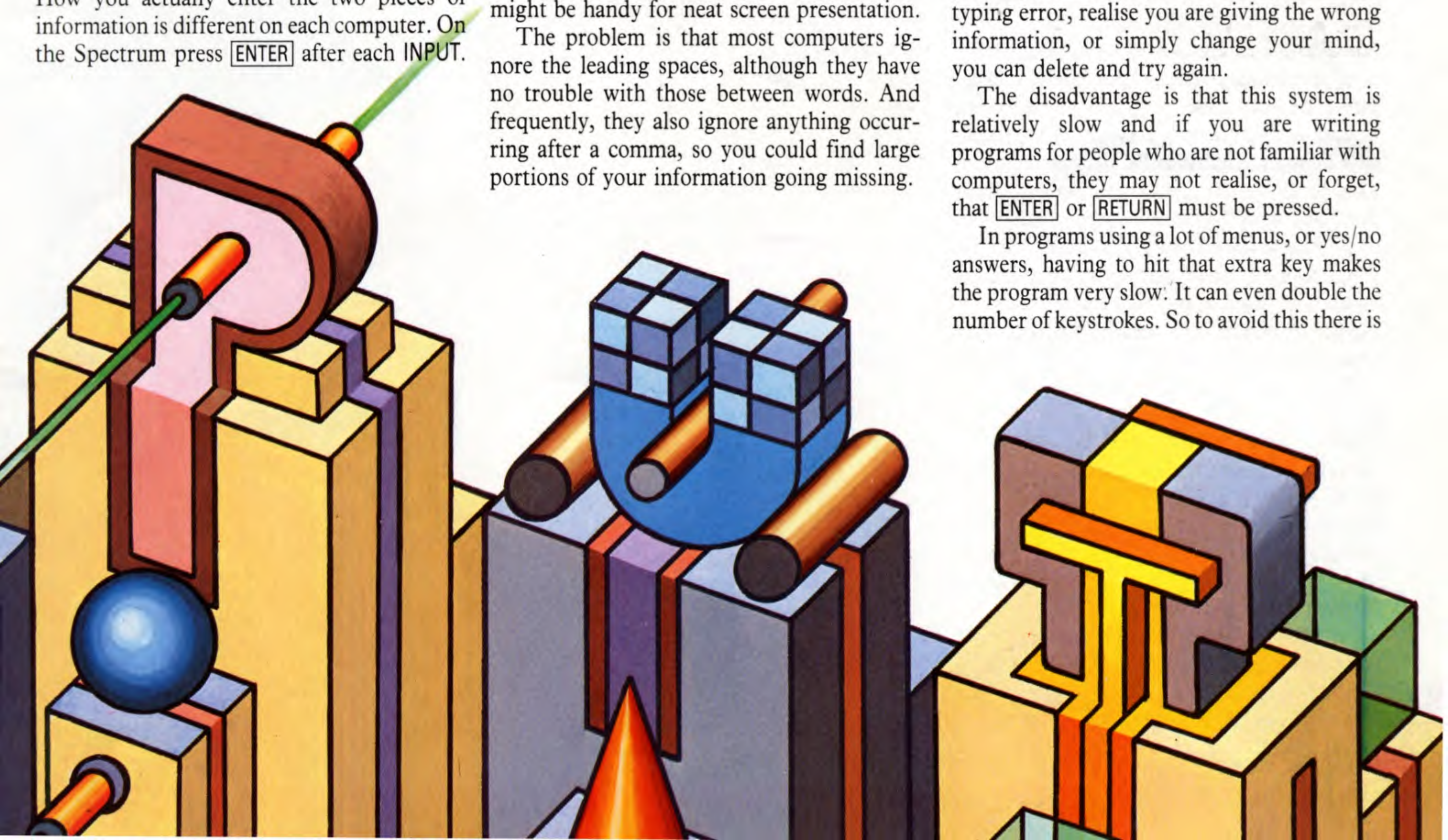
This has the advantage of putting everything up to the next RETURN, including commas and leading spaces, into the variable. It is also more foolproof as you don't have to rely on the user to remember to type in the quotes.

SPEED UP YOUR INPUTS

The main advantage of the INPUT or INPUT LINE command is that you can alter what you are entering right up to the moment of pressing RETURN or ENTER. So if you make a typing error, realise you are giving the wrong information, or simply change your mind, you can delete and try again.

The disadvantage is that this system is relatively slow and if you are writing programs for people who are not familiar with computers, they may not realise, or forget, that ENTER or RETURN must be pressed.

In programs using a lot of menus, or yes/no answers, having to hit that extra key makes the program very slow. It can even double the number of keystrokes. So to avoid this there is



a way of getting the computer to READ which keys you are pressing as you press them.

The command used is INKEY$ on the Sinclairs, Tandy and Dragon, GET$ or INKEY$ on the Acorns and GET on the Commodores.

When the computer encounters one of these keywords, it looks to see if a key is pressed, and then passes the character relating to that key to the variable. This is commonly used for yes/no replies to questions like 'Do you want another game' (see page 35). It is written like this:

```
100 LET A$=GET$
```

```
100 GET A$ : IF A$="""THEN GOTO 100
```

```
100 LET A$=INKEY$ : IF A$="""THEN
   GOTO 100
```

```
100 LET A$=INKEY$
110 IF A$=" " THEN GOTO 100
```

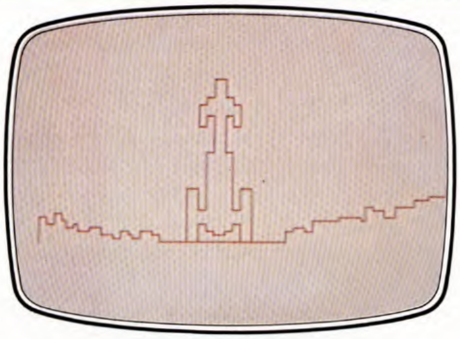
Note that on the Acorn, GET$ makes the computer wait until a key is pressed, while the others have to be made to wait.

Any string variable can be used—A$ is just an example. The computer stops at Line 1Ø0. If you then press R, A$ will contain the letter R. You could enter a number or a space or—any of the characters on the keyboard—even keys such as ENTER. Although the entry can be virtually anything, it can only be one character long. As soon as a key is pressed, the computer carries on.

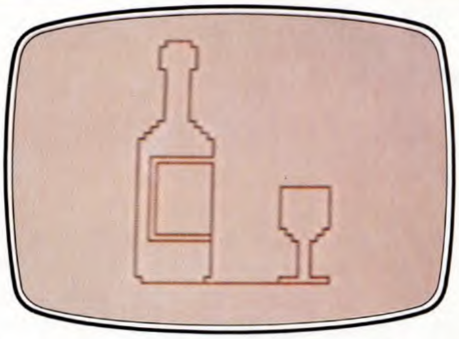
DRAWING ON THE SCREEN

Now look at the following program, which uses four keys to draw on the screen (this program does not RUN on the ZX81).

It uses Z, X, P and L to draw on the Acorn, Dragon and Spectrum, and the cursor keys on the Commodores. You can also press 2 to draw in the screen colour so the line is invisible, and press 1 to return to line drawing again.



You can draw a screen image under complete keyboard control using a short, simple program



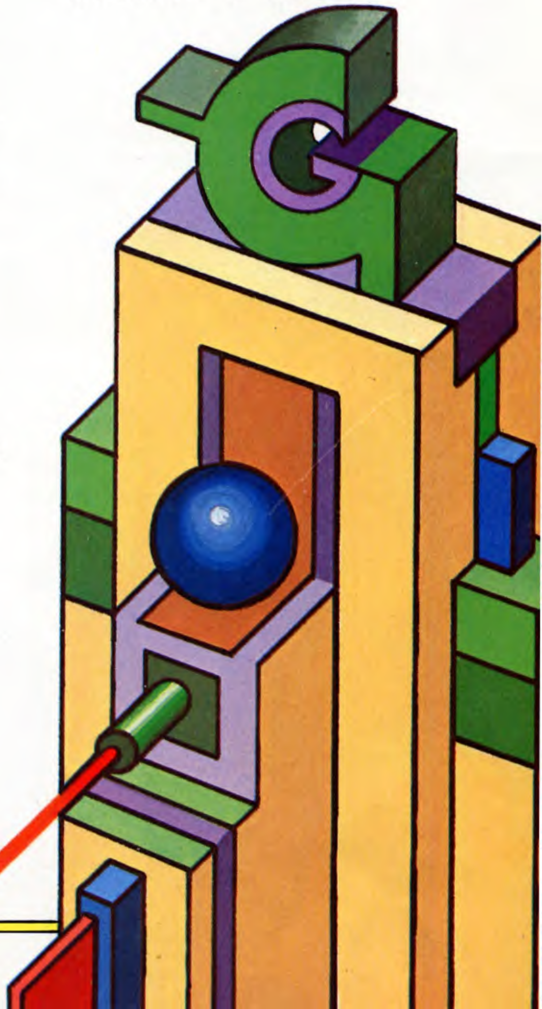
The program allows you to take the line in different directions by pressing particular keys

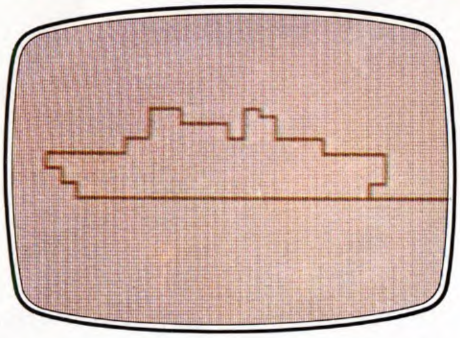
```
10 MODE 1
20 LET X=500: LET Y=500
30 MOVE X,Y
40 REPEAT
50 LET A$=GET$
60 IF A$="P" THEN LET Y=Y+4
70 IF A$="L" THEN LET Y=Y−4
80 IF A$="Z" THEN LET X=X−4
90 IF A$="X" THEN LET X=X+4
100 IF A$="1" THEN GCOLØ,3
110 IF A$="2" THEN GCOLØ,Ø
120 DRAW X,Y
130 UNTIL A$="□"
```

```
10 FOR Z=Ø TO 69:READ X
20 POKE 832+Z,X:NEXT Z :GOTO 90
30 DATA 169,29,141,24,2Ø8,169,59,141,
   17,2Ø8
40 DATA 169,32,133,252,169,Ø,133,251,
   16Ø,Ø,169,Ø,145,251
50 DATA 2ØØ,2Ø8,251,24,165,252,2Ø1,
   63,24Ø,4,23Ø,252,2Ø8,236
60 DATA 162,Ø,169,Ø,157,Ø,64,232,224,
   63,2Ø8,248
70 DATA 162,Ø,169,13,157,Ø,4,157,Ø,5,
   157,Ø,6,157,232,6,232,2Ø8,241,96
90 SYS 832
100 SC=8192:XX=100:YY=100:CO=1
110 LET Y=YY:LET X=XX
120 GET A$:IF A$="" THEN GOTO 120
130 IF A$="▌" THEN LET X=X−1
140 IF A$="▟" THEN LET X=X+1
150 IF A$="◙" THEN LET Y=Y−1
160 IF A$="▙" THEN LET Y=Y+1
165 IF A$="1" THEN CO=1
166 IF A$="2" THEN CO=2
170 LET L=SC+( INT(Y/8)*320+8*
   INT(X/8)+(Y AND 7))
180 IF L<8192 OR L>16191 THEN GOTO
   110
190 LET XX=X:LET YY=Y
200 IFCO=1THEN POKE L, PEEK(L) OR
   (2 ↑ (7−(XX AND 7)))
210 GOTO 110
```

```
10 PMODE4,1:PCLS:SCREEN 1,1
20 LET X=100:LET Y=100
40 LET X1=X:LET Y1=Y
50 LET A$=INKEY$
60 IF A$="P" THEN LET Y=Y1−4
70 IF A$="L" THEN LET Y=Y1+4
80 IF A$="Z" THEN LET X=X1−4
90 IF A$="X" THEN LET X=X1+4
100 IF A$="1" THEN COLOR 5
110 IF A$="2" THEN COLOR Ø
120 IF A$="□" THEN STOP
130 LINE (X,Y)−(X1,Y1),PSET
140 GOTO 40
```

```
10 INK 2
20 PLOT 127,87
30 IF INKEY$="p" THEN DRAW Ø,2
40 IF INKEY$="l" THEN DRAW Ø,−2
50 IF INKEY$="z" THEN DRAW −2,Ø
60 IF INKEY$="x" THEN DRAW 2,Ø
70 IF INKEY$="1" THEN INK 2
```





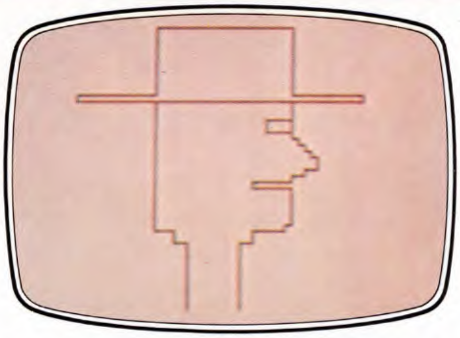
Practise drawing these shapes or any others which you would like to copy following the instructions below

```
8Ø IF INKEY$ = "2" THEN INK 7
9Ø IF INKEY$ = "□" THEN STOP
1ØØ PAUSE 1Ø
11Ø GOTO 3Ø
```

This kind of routine is extremely useful in graphics and games. The line is drawn while the keys are pressed. But note that the computer will take only one key at a time, so diagonal lines are rather difficult to draw as they consist of a series of short steps.

Using INKEY$ or GET$ with single characters is also very handy in any program using menus. The following program PRINTs out a menu as part of a datafile program.

```
1Ø DATA CREATE FILE, ENTER, VIEW,
   EDIT, SEARCH, PRINT, LOAD, SAVE, STOP
15 RESTORE
2Ø FOR N = 1 TO 9
3Ø READ HEADING$
4Ø PRINT TAB(5);N;TAB(1Ø);HEADING$
5Ø NEXT N
6Ø PRINT:PRINT TAB(5)"YOUR CHOICE > "
7Ø LET A$ = GET$
8Ø IF A$ = "1" THEN GOSUB 1ØØØ
9Ø IF A$ = "2" THEN GOSUB 2ØØØ
1ØØ IF A$ = "3" THEN GOSUB 3ØØØ
11Ø IF A$ = "4" THEN GOSUB 4ØØØ
12Ø IF A$ = "5" THEN GOSUB 5ØØØ
13Ø IF A$ = "6" THEN GOSUB 6ØØØ
14Ø IF A$ = "7" THEN GOSUB 7ØØØ
15Ø IF A$ = "8" THEN GOSUB 8ØØØ
16Ø IF A$ = "9" THEN GOSUB 9ØØØ
17Ø GOTO 15
```



Note that diagonals must be drawn as a series of short steps—you cannot press two keys at once

Delete Line 7Ø in the Acorn program above and substitute:

```
7Ø GET A$: IF A$ = "" THEN GOTO 7Ø
```

Delete Line 7Ø above and substitute:

```
7Ø LET A$ = INKEY$: IF A$ = "" THEN
   GOTO 7Ø
```

Delete Lines 1Ø to 7Ø above and substitute:

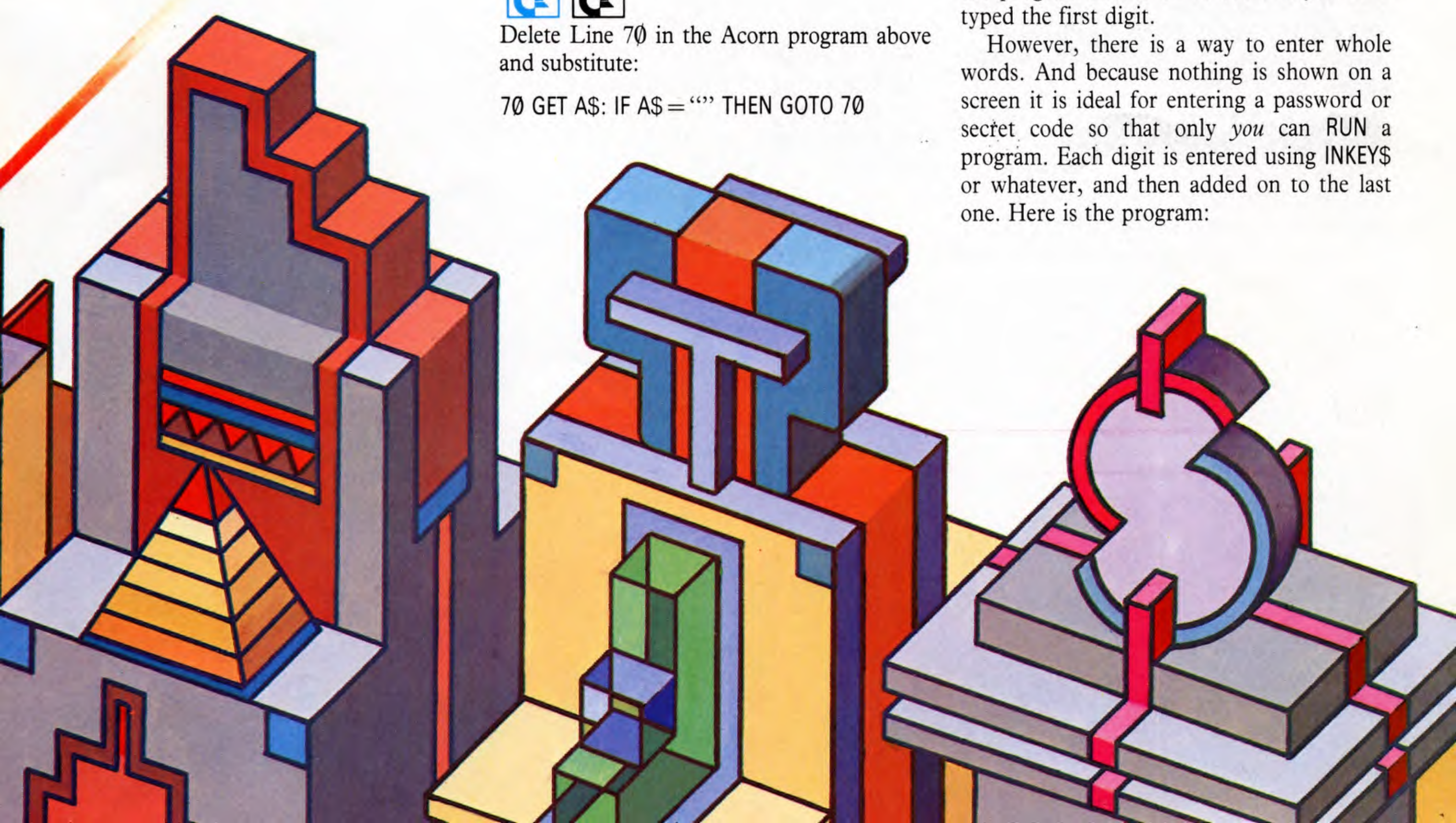
```
1Ø DATA "Create new file", "Enter
   records", "View records", "Edit
   records", "Search records",
   "Print file", "Load file", "Save
   file", "Stop"
15 RESTORE
2Ø FOR n = 1 TO 9
3Ø READ h$
4Ø PRINT TAB 5; n; TAB 1Ø; h$
5Ø NEXT n
6Ø PRINT: PRINT TAB 5; "Your choice > "
7Ø LET A$ = INKEY$ : IF A$ = ""THEN
   GOTO 7Ø
```

With this program, the computer goes straight to the relevant subroutine as soon as you press a key—unless you press a key other than the numbers 1 to 9. In that case, Line 17Ø rePRINTs the menu and then lets you have another go.

A SECRET PASSWORD PROGRAM

The previous program will work perfectly as long as there are no more than nine options. But if you try to type in 1Ø the computer assumes you are entering 1. This is because the program moves on as soon as you have typed the first digit.

However, there is a way to enter whole words. And because nothing is shown on a screen it is ideal for entering a password or secret code so that only *you* can RUN a program. Each digit is entered using INKEY$ or whatever, and then added on to the last one. Here is the program:



```
10 PRINT ''"ENTER PASSWORD"
15 REPEAT
20 LET K$=GET$
30 LET P$=P$+K$
40 UNTIL LEN(P$)=7
50 IF P$<>"BANANAS" THEN END
60 PRINT "O.K."
70 REM (Rest of program follows)
```

```
10 PRINT "♥▣▣▣▣▣▣▣▣"
   TAB(13) "ENTER PASSWORD"
20 FOR Z=1 TO 7
30 GET K$:IF K$="" THEN GOTO 30
40 LET P$=P$+K$
50 NEXT Z
60 IF P$<>"BANANAS" THEN END
70 PRINT "◙" TAB(13) "□PASSWORD
   OKAY□□□"
80 REM (REST OF PROGRAM FOLLOWS)
```

```
10 PRINT "ENTER PASSWORD"
20 LET K$=INKEY$:IF K$="" THEN
   GOTO 20
30 LET P$=P$+K$
40 IF LEN(P$)<>7 THEN GOTO 20
50 IF P$<>"BANANAS" THEN STOP
60 PRINT "O.K."
70 REM (REST OF PROGRAM FOLLOWS)
```

On the ZX81, type entirely in capitals. Omit Line 4∅ and substitute:

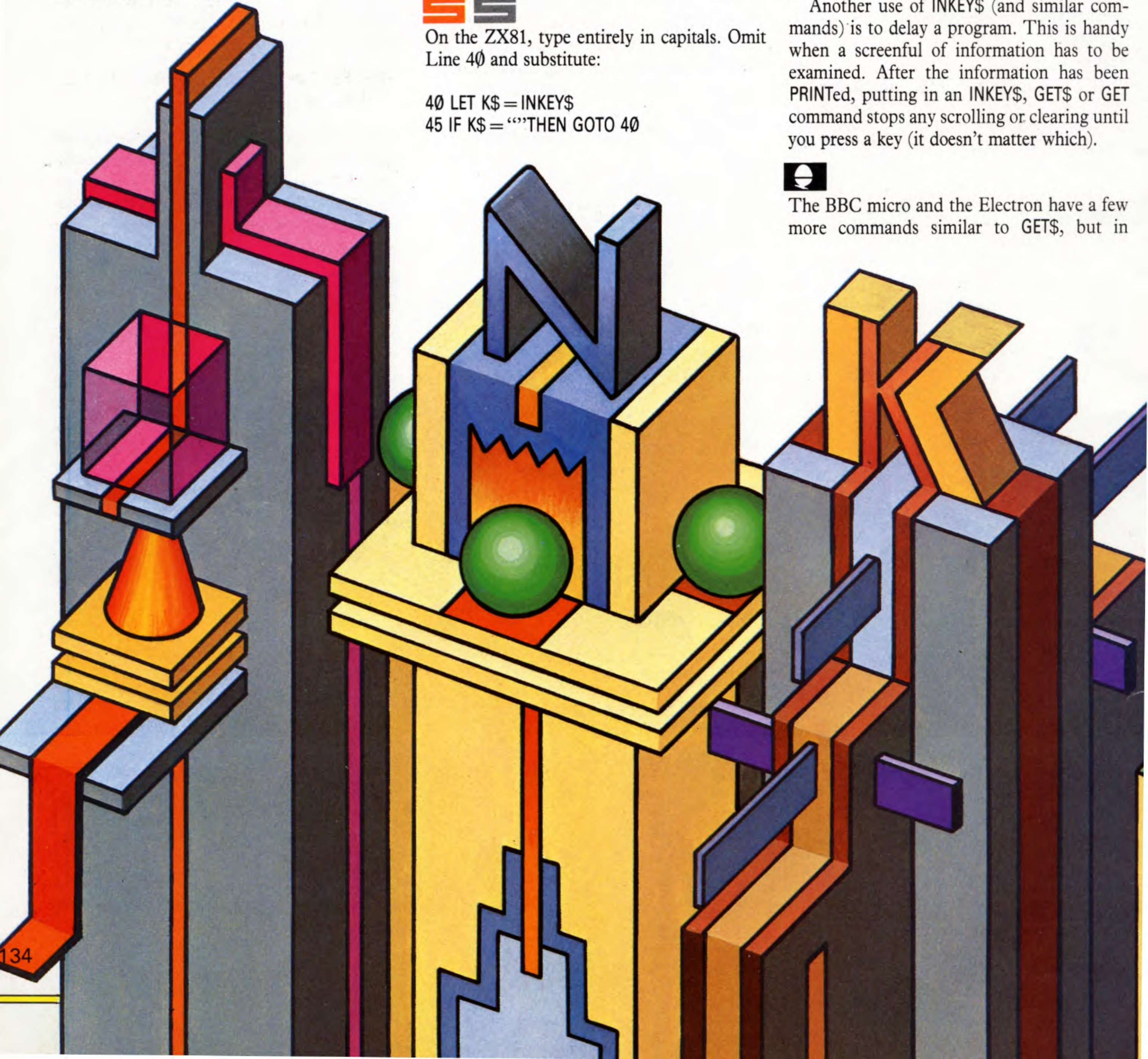
```
40 LET K$=INKEY$
45 IF K$=""THEN GOTO 40
```

```
10 LET p$=""
20 PRINT "ENTER PASSWORD"
30 PAUSE 0
40 LET k$=INKEY$: IF k$="" THEN
   GOTO 40
50 LET p$=p$+k$
60 IF LEN p$<>7 THEN GOTO 30
70 IF p$<>"bananas" THEN STOP
80 PRINT "O.K."
90 REM (rest of program follows)
```

This routine starts off with an empty string (P$) and progressively adds on the characters one by one until the length of the password is correct. Since the characters entered are not PRINTed on the screen, it stops people watching the screen and learning the password.

Another use of INKEY$ (and similar commands) is to delay a program. This is handy when a screenful of information has to be examined. After the information has been PRINTed, putting in an INKEY$, GET$ or GET command stops any scrolling or clearing until you press a key (it doesn't matter which).

The BBC micro and the Electron have a few more commands similar to GET$, but in



certain cases more useful. The closest to GET$ is INKEY$. The only difference is that, whereas a GET$ command makes the computer wait forever, until a key is pressed, with INKEY$ you specify a time limit.

INKEY$ is always followed by a number in brackets. This specifies the waiting time in hundredths of a second. So to make the computer wait for five seconds, you would write: A$=INKEY$ (500). This is an extremely useful way of making a computer pause for a specified length of time.

If a key is pressed before the end of the period, then the computer continues with the program—the time delay is simply a maximum. If no key is pressed before the time is up, then a null string ("") is returned. INKEY$(0) makes the computer scan the keyboard but not wait at all—useful in games where speed is most important.

GET is similar to GET$, but, instead of returning a string, it gives the ASCII number of the key pressed. This does not appear on the screen, but it can be stored in a variable for use by the computer. INKEY is similar to GET but, again, it is the key's ASCII value which is returned. If no key is pressed within the time limit then a value of −1 is given.

INKEY and INKEY$ take their values from the last character in the keyboard buffer. This is a separate section of memory that stores the keypresses. This is not accurate enough or fast enough for many games, where keys are being pressed rapidly or held down as the buffer may be storing a character from earlier in the game. But there is an alternative version which examines the keyboard itself rather than the buffer.

You can follow INKEY with a negative number in brackets. In this case the number is not a time limit, but a special code value of the key you are testing for. So if you wanted to see if the N key was being pressed you would use INKEY (−86). You'll find a list of all the codes in the manual.

INKEY with a negative number checks all keys being pressed, even if you press them at the same time. So this technique is very useful for graphics or games where things like diagonal movement using two keys are needed. The following program demonstrates this. Compare it with the earlier program which used GET$, and you'll see how much smoother the lines are using this method:

```
10 MODE 5
20 X=500:Y=500
30 MOVE X,Y
40 REPEAT
50 IF INKEY(−58) THEN Y=Y+4
60 IF INKEY(−42) THEN Y=Y−4
70 IF INKEY(−26) THEN X=X−4
80 IF INKEY(−122) THEN X=X+4
90 DRAW X,Y
100 UNTIL INKEY(−99)
```

Use the cursor control keys to draw and press the space bar to stop.

At present, this program draws a continuous line and there is no way of leaving a gap. But add these next four lines to the program and have a choice of drawings in black, i.e. invisible, as well as red, yellow or white. Press B, R, Y or W for the colours.

```
42 IF INKEY(−101) THEN GCOL 0,0
44 IF INKEY(−52) THEN GCOL 0,1
46 IF INKEY(−69) THEN GCOL 0,2
48 IF INKEY(−34) THEN GCOL 0,3
```

On Commodore computers there are two keywords, apart from the commonly used INPUT and GET, which can be used to enter information.

INPUT# and GET# are input/output statements normally used to retrieve DATA from a device or file rather than the keyboard. GET# reads a single character at a time, whereas INPUT# retrieves DATA in the form of variables of up to 80 characters in length.

Of these two keywords, INPUT# is probably the more useful. It can be incorporated as part of a useful escape-proof INPUT routine:

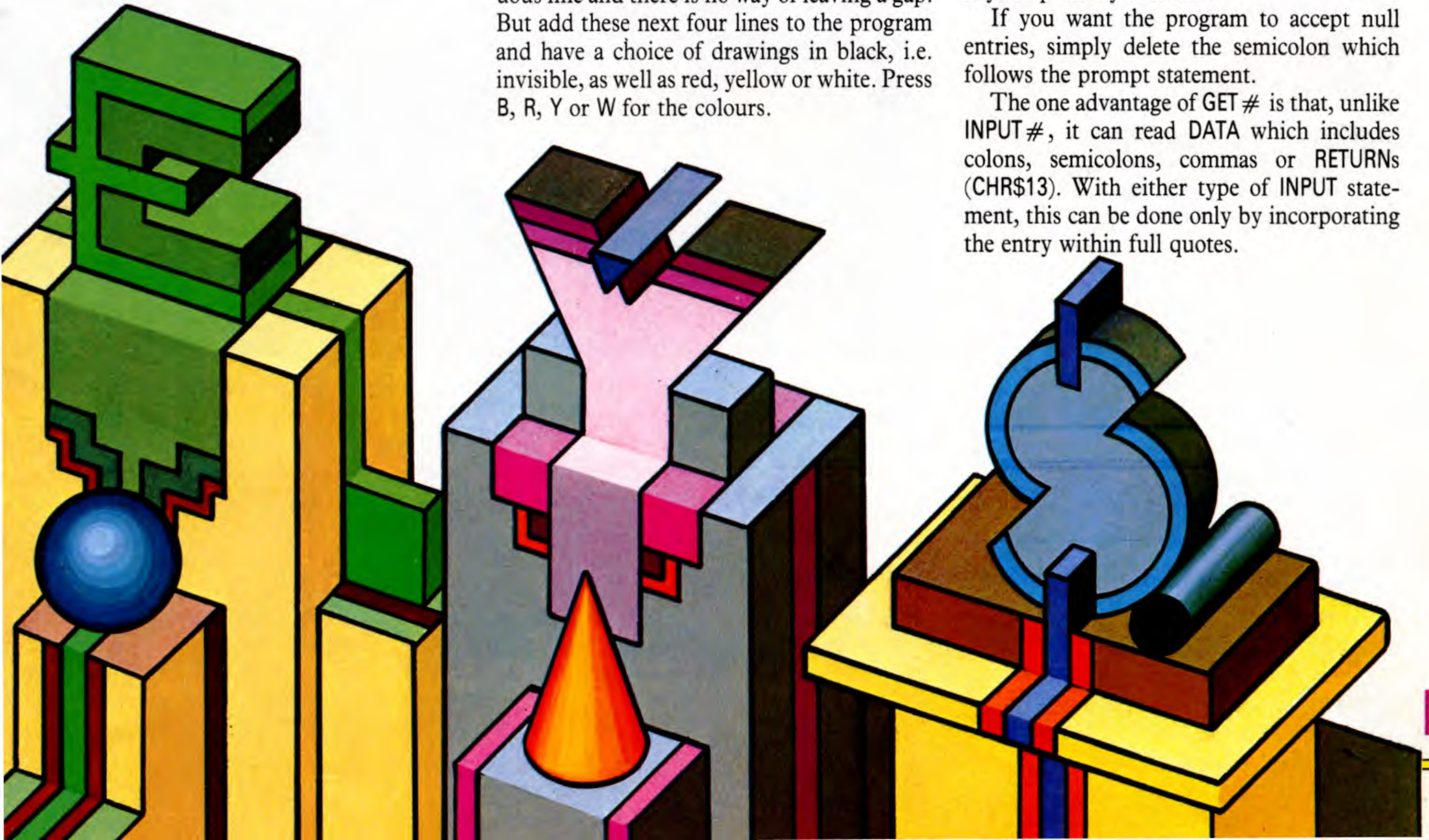
```
100 OPEN 1,0: PRINT "COMMENT/PROMPT";:
    INPUT #1,A$: PRINT: CLOSE1
```

The OPEN and CLOSE statements can in fact span the entire program if necessary: both, however, are essential.

Try RUNning this as a one-line program. Then try to escape by simulating a possible accidental INPUT entry. It's difficult, and points to the usefulness of escape-proof routines such as this when someone perhaps unfamiliar with computers and INPUT routines is left to his or her own devices. You can escape by simultaneously pressing RUN/STOP and RESTORE—an unlikely combination of keys to press by accident.

If you want the program to accept null entries, simply delete the semicolon which follows the prompt statement.

The one advantage of GET# is that, unlike INPUT#, it can read DATA which includes colons, semicolons, commas or RETURNs (CHR$13). With either type of INPUT statement, this can be done only by incorporating the entry within full quotes.



SORT OUT YOUR EXPENSES

Just like a business machine, your computer can store or calculate financial records. Here's a simple program to keep track of your income and expenditure

Keeping track of family spending—'where *does* all the money go?'—is a problem with which most people are only too familiar.

This household accounts program is designed to provide the answers. And it will run on all the machines except the Vic and ZX81.

To update your accounts, you 'feed' it, once a month or whenever else you can spare the time, with the details of your income (for example, from pay slips) and expenditure (for example, from cheque books and standing orders). At any time you like, it will give you an analysis of how your money has been spent, and how your income and expenditure compare for the year to date.

The program itself is quite long. But when you have entered it once, and SAVEd it on one or more tapes, it will last virtually forever—or at least until the tape or tapes wear out.

The program gives you one column for income and seven for expenditure under different headings. These latter subdivisions can of course be varied to suit yourself: all you have to do is alter the wording in the DATA statements in the program when you enter it. Income, however, must be last, and you must have eight 'columns' altogether, or the program will not work.

The program must be SAVEd in two sections—first, the actual program itself, and second, all the information you have fed into it up to your last entry. This means that you will need two program names, one for each of the two bits.

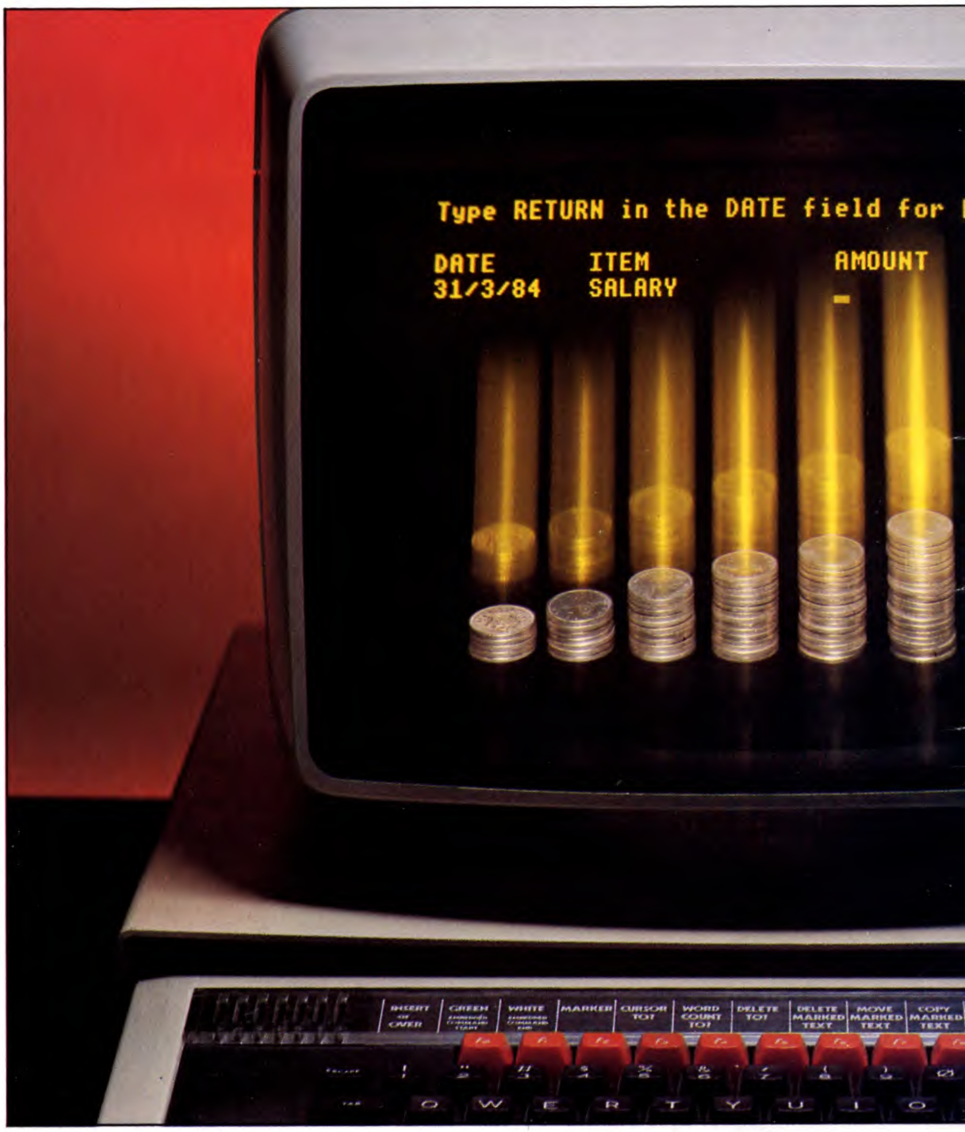
To SAVE the program proper, just follow the normal SAVEing procedure for your machine as given in your manual and/or on pages 22 to 25 of *Input*.

To reLOAD the program, again follow your usual procedure for LOADing taped games or your own programs.

Instructions for SAVEing and LOADing the DATA itself are given below.

When you RUN the program, the main menu will give you seven options:

1 Make an entry
2 View the entries
3 Save on tape
4 Load from tape
5 Printer yes/no
6 Change an entry
7 Quit the program



MAKING AN ENTRY

To make an entry, press 1 when the main menu appears. Do *not* press ENTER or RETURN at this stage.

The computer will ask you in turn for these items of information: Date; Item; Amount; and Category (the category you have already chosen and entered in the DATA statement).

Key in the information in the order given, pressing ENTER or RETURN (whichever your computer uses) after each one.

When you have completed your entries, wait for the computer to ask you for a new date, then press ENTER or RETURN. This will take you back to the main menu.

VIEWING AN ENTRY

To view an entry or series of entries, press key 2 when the main menu appears. Do not press ENTER or RETURN.

The computer will display a table showing the various categories—seven of expenditure, one of income. To select a category, press the appropriate number (again, do not press ENTER or RETURN) and the computer will list

ACORN/BBC: Disc users should delete lines 15, 3Ø and 87Ø and note that there will be less room for entries.

- ENTERING AND SAVEING THE PROGRAM
- THE MENU OPTIONS
- MAKING AN ENTRY IN THE RECORDS
- UPDATING THE RECORDS
- CHECKING THE BALANCE
- TAKING A PRINT-OUT
- STORING YOUR FINANCES ON TAPE

all the items it has in that category, with the total to date at the end.

On the Spectrum, the screen will display the question 'scroll?' if there is insufficient room to display all the entries simultaneously. Do not press N at this stage; you must go right through the listings.

When you have finished, press ENTER or RETURN to get back to the main menu.

If you select option 8, you will get not just an income total, but also the total for all categories of expenditure, plus your balance of income over expenditure (or vice versa).

CHANGING AN ENTRY

When you press 6 for the option to alter an entry, the computer will display a list of all the entries you have made, regardless of category.

You can move backwards or forwards among the list by using the prompts which will appear on the screen. The computer will also tell you how to edit the entry.

Once you have pressed ENTER or RETURN after making the alteration, the computer will automatically return you to the main menu. To make a second alteration, you will have to select option 6 again.

PRINTER OPTION

The printer option command is as simple as a light switch—it is either 'on' or 'off'.

When you press option 5 (without ENTER or RETURN!) on the main menu, the computer will ask you to press Y if you want to use the printer, N if you don't. If you press Y, you will be returned to the main menu and (until you return to option 5 and turn the printer off) all the information that would normally be displayed on the screen when using option 2 will be printed out instead.

Be very careful not to press Y if you have no printer connected. The Spectrum will ignore the instruction in this case, but on any of the other three computers you could lose all the information you have entered so far.

SAVEING AND LOADING

The second stages of SAVEing and LOADing are as follows:

To SAVE the financial DATA you have entered press option 3 (without RETURN or ENTER). Now type in a name for the file—"MONEYFILE", for example. Then press RETURN or ENTER and the 'record' button on your tape recorder. When the DATA has been SAVEd you will be returned to the main menu where you can press option 7 to quit the program.

To LOAD the information that you have SAVEd previously, press option 4 on the main menu. Now enter your file's name and press RETURN or ENTER, then push the 'play' button on your tape recorder. When the program has been loaded the computer will take you back to the main menu.

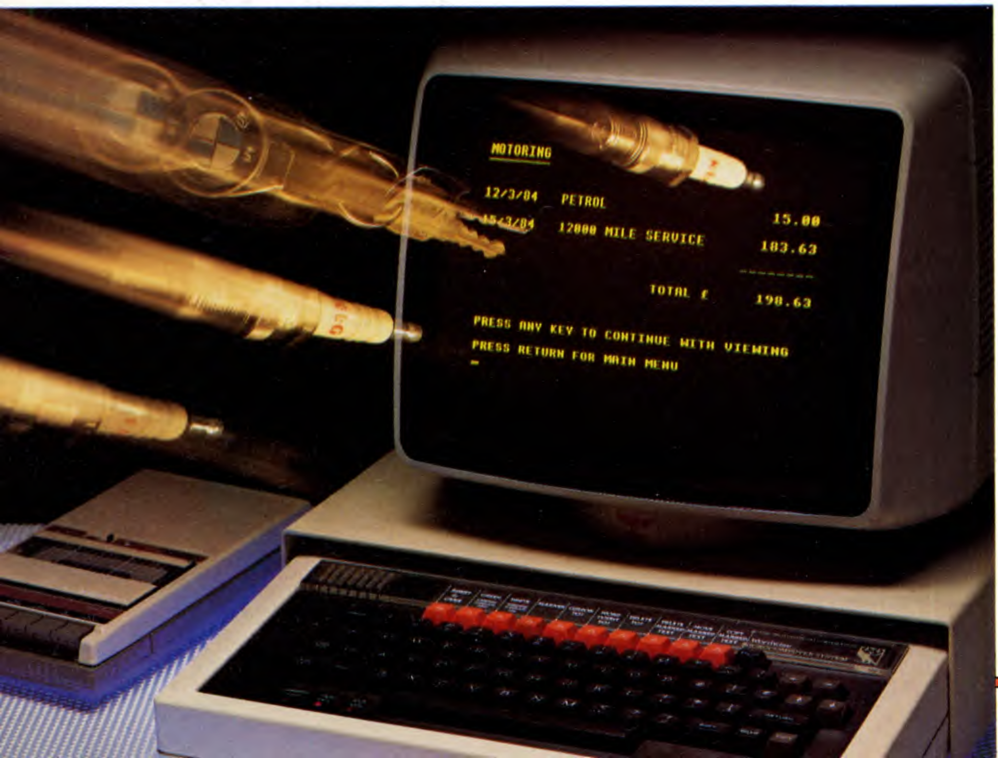
```
10 MODE6
15 * TAPE
20 *OPT1,1
30 *OPT2,1
40 @%=&2020A:N=Ø:W=3:VDU14:
   PAY=Ø: SPENT=Ø:DIM A$(3ØØ),
   A(3ØØ), D$(3ØØ),K$(7)
5Ø FOR T=Ø TO 7:READ K$(T): NEXT T
6Ø PROCMENU
7Ø IF A=1 THEN PROCENTRY
8Ø IF A=2 THEN PROCVIEW
9Ø IF A=3 THEN PROCSAVE
1ØØ IF A=4 THEN PROCLOAD
11Ø IF A=5 THEN PROCPRINT
12Ø IF A=6 THEN PROCCHANGE
13Ø IF A< >7 THEN 6Ø
14Ø PRINT"ARE YOU SURE (Y/N)":
   G=GET AND &5F:IF G< >89 THEN 6Ø
15Ø MODE6:END
16Ø DEF PROCENTRY
17Ø Z=Ø:CLS
18Ø IF N>299 THEN PRINT'"MEMORY
   FULL":SOUND1,-15,1ØØ,5:
   G=INKEY(2ØØ):ENDPROC
19Ø PRINT'"Type RETURN in the DATE
   field for MENU"
2ØØ PRINT'"DATE□□□□□□
   ITEM□□□□□□□□□□□
   □AMOUNT□□□CATGY"
21Ø VDU28,Ø,24,39,4
22Ø INPUT TAB(Ø,Z)D$(N+1):IF
   D$(N+1)="" THEN 35Ø
23Ø INPUT TAB(1Ø,Z)A$(N+1):INPUT
   TAB(26,Z)A(N+1):INPUT TAB(35,Z)
   CA$
24Ø D$(N+1)=LEFT$(D$(N+1),8):
   A$(N+1)=LEFT$(A$(N+1),16)
25Ø GOTO 27Ø
26Ø PRINT TAB(35,Z)"□□□□□":
   INPUT TAB(35,Z)CA$
27Ø X=Ø:FOR T=Ø TO 7:IF INSTR
   (K$(T),CA$)=1 THEN X=X+1:Y=T
28Ø NEXT
29Ø IF X< >1 THEN 26Ø
3ØØ A$(N+1)=CHR$(Y)+A$(N+1)
31Ø IF Y=7 THEN PAY=PAY+A(N+1)
   ELSE SPENT=SPENT+A(N+1)
32Ø Z=Z+1:N=N+1
33Ø IF Z>19 THEN Z=Ø:CLS
34Ø GOTO22Ø
```

```
350 VDU28,Ø,24,39,Ø:ENDPROC
360 DEF PROCSHOWCAT
370 CLS:SUM=Ø:VDU W
380 PRINT'K$(C) 'STRING$(LEN(K$(C)),
   CHR$(224))
390 VDU28,Ø,24,39,3
400 FOR T=1 TO N
410 IF N=Ø THEN 46Ø
420 S$=RIGHT$(A$(T),1)
430 IF ASC(LEFT$(A$(T),1))<>C
   THEN 46Ø
440 PRINT'D$(T)TAB(1Ø)RIGHT$
   (A$(T),LEN(A$(T))-1)TAB(29),
   A(T);
450 SUM=SUM+A(T)
460 NEXT
470 PRINTTAB(32)"---------"
480 PRINTTAB(22)"TOTAL□£",SUM
490 IF C<>7 THEN 52Ø
500 PRINT'"YOUR TOTAL EXPENDITURE
   IS□£";SPENT
510 PRINT''"YOUR BALANCE IS□£";
   PAY-SPENT
520 VDU1,1Ø,1,1Ø
530 VDU 3
540 PRINT''"PRESS ANY KEY TO
   CONTINUE WITH VIEWING"''"PRESS
   RETURN FOR MAIN MENU"
550 G=GET:VDU28,Ø,24,39,Ø:ENDPROC
560 DEF PROCVIEW
570 CLS:PRINT
580 FOR T=Ø TO 7:PRINT'TAB(1Ø);
   STR$(T+1);"□□";K$(T):NEXT
590 PRINT''"WHICH CATEGORY NUMBER ?";
600 C=GET-49
610 IF C=-36 THEN ENDPROC
620 IF C<Ø OR C>7 THEN 6ØØ
630 PROCSHOWCAT
640 IF G=13 THEN 65Ø ELSE 57Ø
650 ENDPROC
660 DEF PROCMENU
670 CLS
680 PRINTTAB(1Ø,2)"MAIN MENU"
690 PRINTTAB(1Ø,5)"1:- Make an entry"
700 PRINTTAB(1Ø,7)"2:- View entries"
710 PRINTTAB(1Ø,9)"3:- Save to tape"
720 PRINTTAB(1Ø,11)"4:- Load from
   tape"
730 PRINTTAB(1Ø,13)"5:- Printer
   option"
740 PRINTTAB(1Ø,15)"6:- Change entry"
750 PRINTTAB(1Ø,17)"7:- Quit program"
760 PRINTTAB(1Ø,2Ø)"SELECT OPTION"
770 A=GET-48
780 IF A<1 OR A>7 THEN 77Ø
790 ENDPROC
800 DEF PROCSAVE
810 INPUT "NAME OF FILE",D$:IF
   D$="" THEN ENDPROC
820 IF D$="" THEN ENDPROC
830 H=OPENOUT(D$):PRINT"SAVING
   INFORMATION NOW":PRINT#H,N
840 FOR T=1 TO N:PRINT#H,D$(T),
   A$(T),A(T):NEXT:CLOSE #H:
   ENDPROC
850 DEF PROCLOAD
860 INPUT"LOAD WHICH FILE",D$
870 PRINT"PRESS PLAY ON RECORDER"
880 H=OPENIN(D$):INPUT#H,N
890 FOR T=1 TO N:INPUT#H,D$(T),
   A$(T),A(T)
900 IF ASC(A$(T))=7 THEN PAY=
   PAY+A(T) ELSE SPENT=
   SPENT+ A(T)
910 NEXT:CLOSE#H:ENDPROC
920 DEF PROCPRINT
930 PRINT"PRINTER (Y/N)"
940 G=GET AND &5F:IF G=89 THEN
   W=2:GOTO 96Ø
950 IF G<>78 THEN 94Ø ELSE W=3
960 ENDPROC
970 DEF PROCCHANGE
980 CLS:T=1
990 IF N=Ø THEN ENDPROC
1000 REPEAT
1010 CLS:PRINT''"ENTRY NUMBER□";
   STR$(T)'D$(T),RIGHT$(A$(T),
   LEN(A$(T))-1)"£";A(T),
   K$(ASC(A$(T)))'
1020 PRINT''"',' TO MOVE BACK"'
   "'.' TO MOVE FORWARD"'"SPACE
   BAR TO CHANGE ENTRY"
1030 A$=GET$
1040 IF A$="," THEN T=T-1: IF T<1
   THEN T=1
1050 IF A$="." THEN T=T+1:IF T>N
   THEN T=N
1060 UNTIL (A$="□" OR A$=CHR$(13))
1070 IF A$=CHR$(13) THEN ENDPROC
1080 E=T:PRINT'"CHANGING THIS
   ENTRY"
1090 CA$=CHR$(ASC(A$(E)))
1100 IF ASC(A$(E))=7 THEN PAY=
   PAY-A(E) ELSE SPENT=SPENT-
   A(E)
1110 INPUT"DATE□",Q$:IF Q$<>""
   THEN D$(E)=Q$
1120 INPUT"ITEM□",Q$:IF Q$<>""
   THEN A$(E)=Q$ ELSE A$(E)=RIGHT$
   (A$(E),LEN(A$(E))-1)
1130 INPUT"AMOUNT□",Q$:IF Q$<>""
   THEN A(E)=EVAL(Q$)
1140 INPUT"CATEGORY□",Q$:IF
   Q$<>"" THEN CA$=Q$
1150 GOTO 117Ø
1160 INPUT"RE-ENTER CATEGORY",CA$
1170 X=Ø:FOR T=Ø TO 7
1180 IF INSTR(K$(T),CA$)=1 THEN
   X=X+1:Y=T
1190 NEXT T
1200 IF X<>1 THEN 116Ø
1210 A$(E)=CHR$(Y)+A$(E)
1220 IF Y=7 THEN PAY=PAY+A(E) ELSE
   SPENT=SPENT+A(E)
1230 PRINT"CORRECTION MADE":
   G=INKEY(2ØØ)
1240 ENDPROC
1250 DATA HOUSEKEEPING,
   ENTERTAINMENT, RATES & RENT,
   CLOTHING, MOTORING, HOLIDAYS,
   MISCELLANEOUS, INCOME
```

The □ symbol denotes an important space. Enter on the space key, not as a graphic.

```
50 LET mn=2ØØ: IF PEEK 23733=127
   THEN LET mn=1ØØ
1ØØ DIM c$(8,16): DIM a(mn): DIM
   a$(mn,23)
11Ø LET u=Ø: LET v=1
12Ø FOR n=v TO 8: READ c$(n): NEXT n
13Ø POKE 23658,8
14Ø LET k$=".ØØ": FOR n=v TO 7:
   LET k$=k$+CHR$ 8: NEXT n
19Ø LET p=2: LET tt=u: LET cr=u
2ØØ CLS : PRINT BRIGHT v; PAPER
   2; INK 6;AT 2,6; "□□M□A□I□N□
   □M□E□N□U□□"
21Ø PRINT BRIGHT v; PAPER 7;AT
   5,6;"□1:–□MAKE AN ENTRY□□";
   AT 7,6;"□2:–□VIEW
   ENTRIES□□□";AT 9,6;"□3:
   –□SAVE TO TAPE□□□"; AT
   11,6;"□4:–□LOAD FROM TAPE□";
   AT 13,6;"□5:–□PRINTER OPTION□";
   AT 15,6;"□6:–□CHANGE
   ENTRY□□□";AT 17,6;"□7:
   –□QUIT PROGRAM□□□"
22Ø PRINT INK 3; FLASH v; BRIGHT
   v;AT 2Ø,6;"□–□SELECT
   OPTION.□–□"
23Ø IF INKEY$="" THEN GOTO 23Ø
24Ø LET z$=INKEY$: IF z$<"1" OR
   z$>"7" THEN GOTO 23Ø
25Ø CLS : GOSUB 1ØØØ*VAL z$
26Ø GOTO 2ØØ
1ØØØ LET c=u
1ØØ5 LET c=c+v: IF c=mn+v THEN
   RETURN
1ØØ6 IF a$(c,v)="□" THEN GOTO 1Ø1Ø
1ØØ7 GOTO 1ØØ5
1Ø1Ø PRINT AT u,u; BRIGHT v; PAPER
   2; INK 7;"□□DATE□□□□□□
   ITEM□□□□□□AMOUNT□CAT"
1Ø15 IF c=mn+v THEN RETURN
1Ø2Ø INPUT "Enter date□"; LINE
   a$(c,2 TO 9): IF a$(c,2)="□"
   THEN RETURN
1Ø3Ø PRINT TAB u;a$(c,2 TO 9);
1Ø4Ø INPUT "Enter item□"; LINE
   a$(c,1Ø TO 23): IF a$(c,1Ø)="□"
   THEN GOTO 1Ø4Ø
1Ø5Ø PRINT TAB 9;a$(c,1Ø TO 21);
1Ø6Ø INPUT "Amount□";a(c): IF
   a(c)=u THEN GOTO 1Ø6Ø
1Ø7Ø LET w=a(c)*1ØØ: LET v$=STR$
   w: PRINT TAB 27–LEN v$;a(c);
1Ø8Ø INPUT "Category□"; LINE f$:
   IF f$="" THEN GOTO 1Ø8Ø
1Ø9Ø FOR n=v TO 8: IF f$=c$(n,v
   TO LEN f$) THEN GOTO 113Ø
11ØØ NEXT n: GOTO 1Ø8Ø
113Ø IF n=8 THEN LET cr=cr+a(c)
114Ø IF n<>8 THEN LET tt=tt+a(c)
115Ø PRINT TAB 29;c$(n,v TO 3)
116Ø LET a$(c,v)=CHR$ (48+n)
12ØØ LET c=c+v: GOTO 1Ø15
2ØØØ FOR n=v TO 8: PRINT PAPER v;
   INK 7;AT n*2,6;"□";n;":–□";
   c$(n): NEXT n
2Ø1Ø PRINT FLASH v; INK 2;AT 19,3;
   "□Select category (1 to 8)□"
2Ø2Ø IF INKEY$="" THEN GOTO 2Ø2Ø
2Ø3Ø LET z$=INKEY$: IF z$<"1" OR
   z$>"8" THEN GOTO 2Ø2Ø
2Ø4Ø LET t=u: LET c=u
2Ø5Ø CLS : PRINT #p; PAPER 6;
   BRIGHT v;TAB 1Ø;c$(VAL z$);
   TAB 31; "□"
2Ø55 LET c=c+v: IF c=mn THEN GOTO
   25ØØ
2Ø6Ø IF a$(c,v)="□" THEN GOTO 25ØØ
2Ø7Ø IF a$(c,v)<>z$ THEN GOTO 2Ø55
2Ø8Ø PRINT #p;a$(c,2 TO 9); TAB 1Ø;
   a$(c,1Ø TO 23);
2Ø9Ø LET am=a(c)*1ØØ: LET n$=STR$
   am: PRINT #p;TAB 29;k$;TAB 31–
   LEN n$;a(c)
21ØØ LET t=t+a(c)
211Ø GOTO 2Ø55
25ØØ PRINT #p;TAB 25;
   "–––––––": LET tx=t*1ØØ:
   LET n$=STR$ tx: PRINT #p;TAB
   12;"TOTAL:–□";TAB 29;k$;TAB
   31–LEN n$;t
251Ø IF z$<>"8" THEN GOTO 259Ø
252Ø LET tz=tt*1ØØ: LET n$=STR$
   tz: PRINT '#p;"TOTAL
   EXPENDITURE:–□";TAB 29;k$;TAB
   31–LEN n$;tt
253Ø LET ba=(t–tt)*1ØØ: LET n$=
   STR$ ba: PRINT '#p;TAB 1Ø;
   "BALANCE:–□";TAB 29;k$;TAB 31–
   LEN n$;ba/1ØØ
259Ø PRINT PAPER 2; INK 7'
   "□□□□Press a key to
   continue□□□□"
26ØØ PAUSE u: IF PEEK 2356Ø=13
   THEN RETURN
261Ø CLS : GOTO 2ØØØ
3ØØØ GOSUB 8ØØØ: IF re=v THEN
   RETURN
3Ø1Ø PRINT PAPER 6;AT 1Ø,u;"□Enter
   a file name for the data□":
   INPUT LINE w$: IF LEN w$>1Ø OR
   LEN w$<v THEN GOTO 3Ø1Ø
3Ø2Ø CLS : SAVE w$ DATA a(): SAVE
   w$ DATA a$(): RETURN
4ØØØ GOSUB 8ØØØ: IF re=v THEN
   RETURN
4Ø1Ø PRINT BRIGHT v;AT 1Ø,u;"Enter
   name of data to be loaded":
   INPUT LINE w$: IF LEN w$>1Ø
   THEN GOTO 4Ø1Ø
4Ø2Ø PRINT PAPER 3; INK 7;AT 1Ø,u;
   "□□□Insert tape and press
   play □□□"
4Ø3Ø LOAD w$ DATA a(): LOAD w$
   DATA a$()
4Ø4Ø LET cr=u: LET tt=u: FOR n=v
   TO mn: IF a$(n,v)="8" THEN LET
   cr=cr+a(n)
4Ø5Ø IF a$(n,1)<>"8" THEN LET
   tt=tt+a(n)
4Ø6Ø NEXT n: RETURN
```

```
5000 PRINT BRIGHT v;AT 10,u;
   "□ Do you want to print out Y/N?□"
5010 PAUSE u: IF INKEY$ = "" THEN
   GOTO 5010
5020 LET z$ = INKEY$
5030 IF z$ = "N" THEN LET p = 2:
   RETURN
5040 IF z$ = "Y" THEN LET p = 3:
   RETURN
5050 GOTO 5010
6000 LET c = v: IF a(c) = u THEN RETURN
6010 PRINT AT u,u; BRIGHT v;
   PAPER (VAL a$(c,v)) - v; INK 9;
   "□Number□";c,c$(VAL a$(c,v))
6015 PRINT PAPER 2; INK 7;'"DATE
   □□□□□□ITEM□□□□□
   □□□□□□AMOUNT□":
   PRINT 'a$(c,2 TO 9);TAB 10;a$
   (c, 10 TO 23);
6020 LET am = a(c)*100: LET n$ = STR$
   am: PRINT TAB 29;k$;TAB 31 - LEN
   n$;a(c)
6030 PRINT PAPER 3; INK 7;AT 20,u;
   "□A□ - □Forwards□□□□
   Q□ - □Backwards□□□□□□□
   EDIT to alter record
   □□□□□□□"
6040 PAUSE u
6050 IF INKEY$ = "Q" AND c > v THEN
   LET c = c - v: GOTO 6010
6060 IF INKEY$ = "A" AND c < > mn THEN
   LET c = c + v
6070 IF a(c) = u THEN LET c = c - v
6080 IF PEEK 23560 = 7 THEN GOTO 6100
6090 GOTO 6010
6100 INPUT BRIGHT v; "Enter new
   date□"; LINE a$(c,2 TO 9): IF
   a$(c,2) = "□" THEN GOTO 6100
6110 PRINT AT 5,u;a$(c,2 TO 9)
6120 INPUT BRIGHT v;"Enter new
   item□"; LINE a$(c,10 TO 23): IF a$
   (c,10) = "□" THEN GOTO 6120
6130 PRINT AT 5,10;a$(c,10 TO 23)
6135 IF a$(c,v) = "8" THEN LET
   cr = cr - a(c)
6136 IF a$(c,v) < > "8" THEN LET
   tt = tt - a(c)
6140 INPUT BRIGHT v;"Enter new
   amount□";a(c): IF a(c) = u THEN
   GOTO 6140
6150 LET am = a(c)*100: LET n$ = STR$
   am: PRINT AT 5,29;k$;TAB 31 - LEN
   n$;a(c)
6160 INPUT BRIGHT v;"Enter new
   category□"; LINE f$: IF f$ = ""
   THEN GOTO 6160
6170 FOR n = v TO 8: IF f$ = c$(n,v
   TO LEN f$) THEN GOTO 6190
6180 NEXT n: GOTO 6160
6190 LET a$(c,v) = CHR$ (48 + n)
6200 IF n = 8 THEN LET cr = cr + a(c)
6210 IF n < 8 THEN LET tt = tt + a(c)
6220 RETURN
7000 GOSUB 8000: IF re = v
   THEN RETURN
7010 RANDOMIZE USR u
8000 PRINT PAPER 4;AT 10,9;"□Are
   you sure?□"
8010 PAUSE u: LET re = u: IF INKEY$
   < > "Y" THEN LET re = v
8020 RETURN
9000 DATA "HOUSEKEEPING",
   "ENTERTAINMENT","RENT AND
   RATES","CLOTHING","MOTORING",
   "HOLIDAYS","MISCELLANEOUS",
   "INCOME"
```



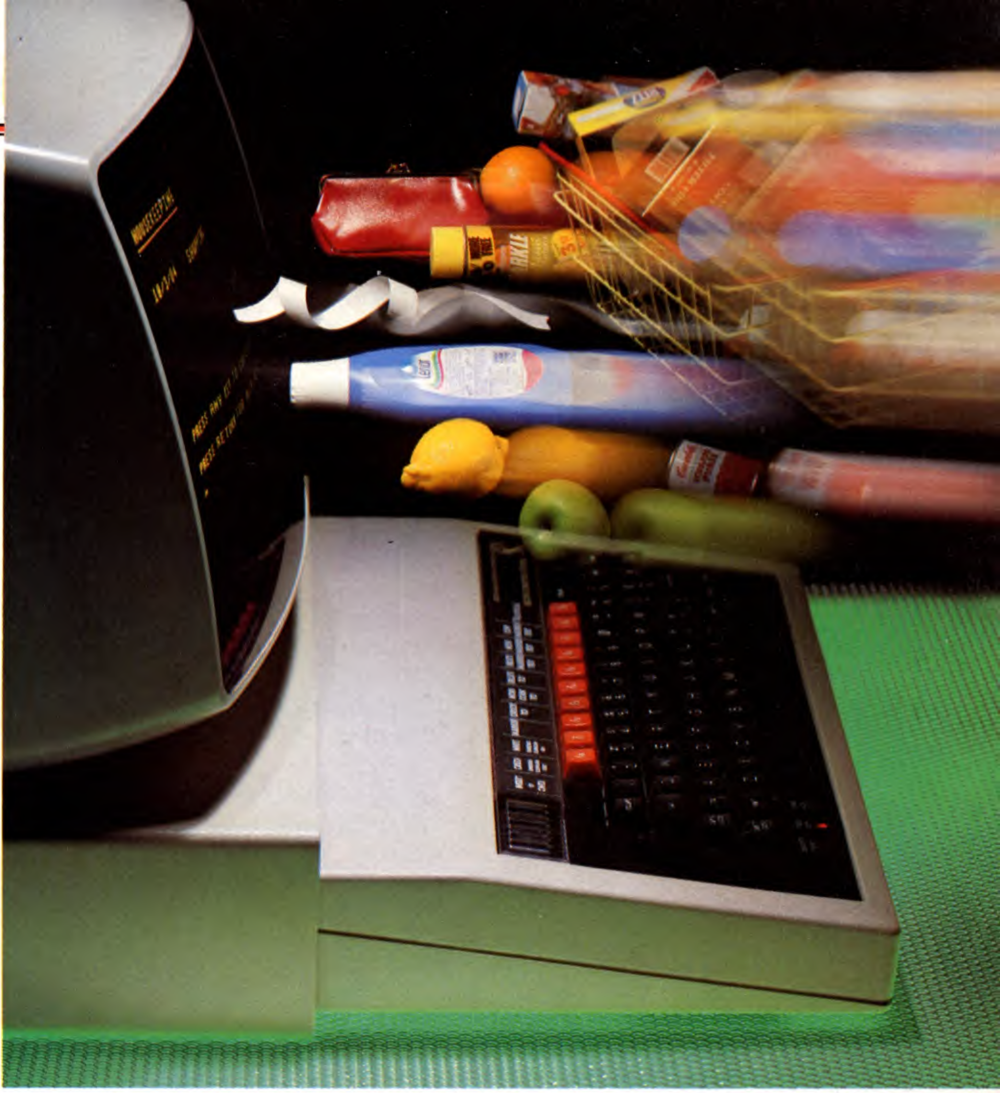
```
MOTORING

12/3/84   PETROL                       15.00
15/3/84   12000 MILE SERVICE          183.63
                                   ---------
                       TOTAL £        198.63

PRESS ANY KEY TO CONTINUE WITH VIEWING
PRESS RETURN FOR MAIN MENU
_
```

On the Tandy, use 247 instead of 223 in Lines 6040, 6050 and 6060.

```
10 PMODE0:PCLEAR1:CLEAR10000
20 DIM TE$(200),AM(200),DA$(200),
   CT$(8),CA(200)
30 FOR N = 1 TO 8:READ CT$(N):NEXT
40 DATA HOUSEKEEPING,ENTERTAINMENT,
   RENT AND RATES,CLOTHING,MOTORING,
   HOLIDAYS,MISCELLANEOUS,INCOME
50 U1$ = "#######.##":
   U2$ = "####.##"
60 CLS4:PRINT@11,"main menu";:
   PRINT@70,"1: - □ENTER DATA
   □□□□□□";:PRINT@134,
   "2: - □VIEW ENTRIES□□□□";:
   PRINT@198,"3: - □SAVE TO
   TAPE□□□□";
70 PRINT@262,"4: - □LOAD FROM TAPE
   □□";:PRINT@326,"5: - □PRINTER
   OPTION□□";:PRINT@390,"6: - □
   CHANGE AN ENTRY□";:PRINT@454,
   "7: - □QUIT PROGRAM□□□□";
80 A$ = INKEY$:IF A$ < "1" OR A$ > "7"
   THEN 80
90 ON VAL(A$) GOSUB 1000,2000,3000,
   4000,5000,6000,7000
100 GOTO 60
1000 CLS:IF NU > 200 THEN PRINT@264,
   "MEMORY FULL□!":PLAY"T10ABCDEFG
   P1P1":RETURN
1010 GOSUB 1160
1020 GOSUB 1250:INPUT"DATE□";
   DA$(NU)
1030 IF DA$(NU) = "" THEN RETURN
1040 IF LEN(DA$(NU)) > 8 THEN 1020
1050 PRINT@L,DA$(NU);
1060 GOSUB 1250:LINEINPUT"ITEM□?";
   TE$(NU):IF LEN(TE$(NU)) > 25 THEN
   1060
1070 A$ = LEFT$(TE$(NU),11):PRINT@L +
   15 - LEN(A$)/2,A$;
1080 GOSUB 1250:INPUT"AMOUNT□";A
1090 IF A > 9999.99 OR A < 0 THEN 1080
1100 PRINT@L + 21,USING U2$;A;:
   AM(NU) = A
1110 GOSUB 1250:INPUT"CATEGORY□";
   CA$
1120 GOSUB 1180:IF F = 0 THEN 1110
1130 CA(NU) = NM:PRINT@L + 29,LEFT$
   (CT$(CA(NU)),3);
1140 IF NM < > 8 THEN GT = GT + A
1150 NU = NU + 1:L = L + 32:IF L = 448 THEN
   1000 ELSE 1020
1160 L = 64:PRINT@2,"date"TAB(13)"item"
   TAB(22)"amount"TAB(29)"cat";
1170 RETURN
1180 IF VAL(CA$) < > 0 THEN 1230
1190 F = 0:FOR N = 1 TO 8
```



```
1200 IF CA$=LEFT$(CT$(N),LEN(CA$))
   THEN F=F+1:NM=N
1210 NEXT:IF F>1 THEN F=Ø
1220 RETURN
1230 IF VAL(CA$)>8 THEN F=Ø:RETURN
1240 NM=VAL(CA$):F=1:RETURN
1250 PRINT@448,"□":PRINT@449,;:RETURN
2000 CLS3:FOR N=1 TO 8
2010 PRINT@69+N*32,N;MID$(":-□"+
   CT$(N)+STRING$(12,"□"),1,20);
2020 NEXT
2030 TT=Ø:PRINT@449,"WHICH
   CATEGORY□?";
2040 A$=INKEY$:IF A$<"1" OR A$>"8"
   THEN 2040
2050 NM=VAL(A$)
2060 IF PT=1 THEN PRINT#-2,CHR$(13):
   PRINT#-2,TAB(21-LEN(CT$(NM))/2);
   CT$(NM):PRINT#-2,"□□DATE"TAB
   (20)"ITEM"TAB(39)"AMOUNT"
2070 GOSUB 2280
2080 FOR NN=Ø TO NU
2090 IF CA(NN)<>NM THEN 2150
2100 IF PT=1 THEN PRINT#-2,USING F$;
   DA$(NN);TE$(NN);AM(NN)
2110 PRINT@L,DA$(NN);:A$=LEFT$(TE$
   (NN),15):PRINT@L+17-LEN(A$)/2,A$;:
   PRINT@L+25,USING U2$;AM(NN);:
   TT=TT+AM(NN)
2120 L=L+32:IF (L=448 AND NM<>8)
   OR (L=352 AND NM=8) THEN PRINT
   @465, "scroll□?"; ELSE GOTO 2150
2130 A$=INKEY$:IF A$="" THEN 2130
2140 GOSUB 2280
2150 NEXT:IF PT=1 THEN PRINT#-2,
   CHR$(13):IF NM<>8 THEN PRINT
   #-2,TAB(28);:PRINT#-2,USING
   "TOTAL□="+U1$;TT
2160 PRINT@463,USING"total□="+U1$;TT;
2170 IF NM<>8 THEN 2250
2180 IF PT=Ø THEN 2220
2190 PRINT#-2,TAB(21);:PRINT#-2,
   USING"TOTAL INCOME□="+U1$;TT
2200 PRINT#-2,TAB(16);:PRINT#-2,
   USING"TOTAL EXPENDITURE□="
   +U1$;GT:PRINT#-2,TAB(35)
   "––––––––––"
2210 PRINT#-2,TAB(26);:PRINT#-2,
   USING"BALANCE□="+U1$;TT-GT
2220 PRINT@392,USING"total income□="
   +U1$;TT;
2230 PRINT@419,USING"total
   expenditure□="+U1$;GT;
2240 PRINT@461,USING"balance□="+
   U1$; TT-GT;
2250 A$=INKEY$:IF A$="" THEN 2250
2260 IF A$<>CHR$(13) THEN 2000
2270 RETURN
2280 L=64:CLS NM:PRINT@(33-LEN(CT$
   (NM)))/2,CT$(NM);
2290 PRINT @34,"date";:PRINT@46,
   "item";:PRINT@57,"amount";
2300 FOR N=32 TO 416 STEP 32
2310 POKE N+1032,122+NM*16:POKE
   N+1048,117+16*NM
2320 NEXT:RETURN
3000 CLS:MOTORON:PRINT@65,"POSITION
   TAPE THEN PRESS [ENTER]"
3010 A$=INKEY$:IF A$="" THEN 3010
3020 MOTOROFF:PRINT@65,"PRESS
   RECORD ON TAPE □□□□□□
   □□□□□□ THEN PRESS [ENTER]"
3030 A$=INKEY$:IF A$="" THEN 3030
3040 CLS:PRINT@65,;:INPUT"NAME OF
   DATA□";DA$
3050 OPEN "O",#-1,DA$
3060 PRINT#-1,NU
3070 FOR N=Ø TO NU-1
3080 PRINT#-1,DA$(N),TE$(N),AM(N),
   CA(N)
3090 NEXT:CLOSE #-1:RETURN
4000 CLS:PRINT@65,;:INPUT"FILE
   NAME";DA$
4010 MOTORON:PRINT@64, "POSITION
   TAPE THEN PRESS [ENTER]"
4020 A$=INKEY$:IF A$="" THEN 4020
4030 MOTOROFF:GT=Ø:OPEN "I",
   #-1,DA$
4040 PRINT@129,"FOUND□";DA$
4050 INPUT#-1,NU
4060 FOR N=Ø TO NU-1
4070 INPUT#-1,DA$(N),TE$(N),AM(N),
   CA(N)
4080 IF CA(N)<>8 THEN GT=GT+AM(N)
4090 NEXT:CLOSE #-1:RETURN
5000 CLS:PRINT@65,"DO YOU WANT THE
   PRINTER ON□?□□□□□(Y/N)";
5010 A$=INKEY$:IF A$<>"Y" AND
   A$<>"N" THEN 5010
5020 PRINT" OK":IF A$="N" THEN PT=
   Ø:RETURN
5030 F$="%□□□□□□□□
   %%□□□□□□□□□□
   □□□□□□□□□□□□□
   %□□□"+U2$:PT=1:RETURN
6000 IF NU=Ø THEN RETURN
6010 CLS:PRINT"□□date"TAB(12)
   "item"TAB(22)"amount"TAB(29)"cat"
6020 PRINT@417,"PRESS [DOWN] TO GO
   FORWARD□□□□□□OR
   [UP] TO GO BACKWARDS."
6030 PRINT@481,"PRESS THE SPACE BAR
   TO EDIT.";:M=Ø:GOTO 6080
6040 IF PEEK (341)=223 AND M>Ø THEN
   M=M-1:GOTO 6080
6050 IF PEEK(342)=223 AND M<NU-1
   THEN M=M+1:GOTO 6080
```

```
6060 IF PEEK(345)=223 THEN 6100
6070 GOTO 6040
6080 PRINT@64,USING"%□□□□□□
    %□%□□□□□□□□□□□
    %####.##□%□□%";
    DA$(M);LEFT$(TE$(M),11);AM(M);
    LEFT$(CT$(CA(M)),3)
6090 GOTO 6040
6100 IF CA(M)<>8 THEN
    GT=GT−AM(M)
6110 INPUT"NEW DATE□";D$:IF D$=""
    THEN 6130
6120 IF LEN(D$)>8 THEN 6110 ELSE
    DA$(M)=D$
6130 INPUT"NEW ITEM□";D$:IF D$=""
    THEN 6150
6140 TE$(M)=D$
6150 INPUT"NEW AMOUNT□";A:IF A=0
    THEN 6170
6160 IF A<0 OR A>9999.99 THEN 6150
    ELSE AM(M)=A
6170 INPUT"NEW CATEGORY□";CA$:IF
    CA$="" THEN 6200
6180 GOSUB1180:IF F=0 THEN 6170
6190 CA(M)=NM
6200 IF CA(M)<>8 THEN
    GT=GT+AM(M)
6210 RETURN
7000 CLS:PRINT@69,"ARE YOU SURE□
    (Y/N)□?";
7010 A$=INKEY$:IF A$<>"Y" AND
    A$<>"N" THEN 7010
7020 IF A$="N" THEN RETURN
```

Q+A

Can the Dragon and Tandy produce lower case characters—such as the letters 'a' to 'z'?

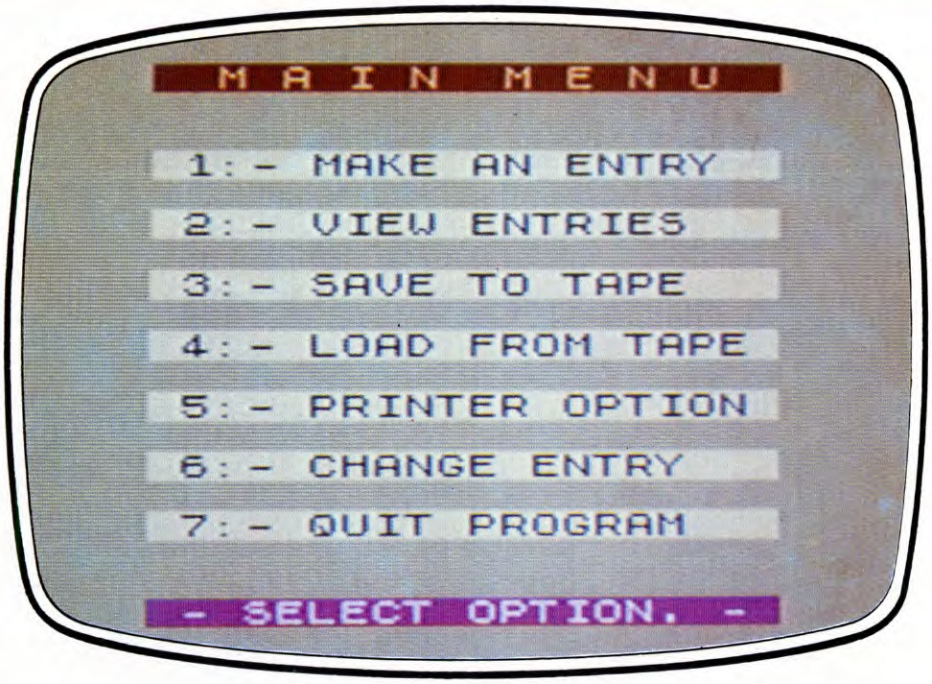
The Dragon can output lower case characters on a printer, but not on the screen. This is good news if you have a printer, because you can print documents, such as letters or broadsheets, more attractively using a mixture of the cases. But even if you use only a TV set, you can still distinguish between the cases.

When you switch on the machine, it is set up automatically in upper case mode—it gives black capitals on a green background. To enter lower case mode, press SHIFT and Ø together. This releases a CAPS lock, so that when the letters from 'A' to 'Z' are pressed, they appear in reversed colours—green capitals on a black background.

All commands to the computer must be in upper case—'A', for example. A printer outputs 'A' as upper case 'A' and reversed 'A' as lower case 'a'. To return to upper case mode, press SHIFT and Ø together again.

```
20 PRINT "♥":POKE 53280,0:POKE
    53281,0:DIM D$(4,400):CO=0
30 A$(1)="HOUSEKEEPING":A$(2)=
    "ENTERTAINMENT":A$(3)="RENT &
    RATES"
40 A$(8)="INCOME":A$(4)="CLOTHING":
    A$(5)="MOTORING":A$(6)="HOLIDAY"
50 A$(7)="MISCELLANEOUS"
60 PRINT "♥π■■■"TAB
    (13)"■□□□□MAIN
    MENU□□□□":PRINT TAB(13)
    "†■■1. MAKE AN ENTRY"
70 PRINT TAB(13)"■2. VIEW ENTRIES":
    PRINT TAB(13)"■3. SAVE TO TAPE"
80 PRINT TAB(13)"■4. LOAD FROM
    TAPE":PRINT TAB(13)"■5. PRINTER
    OPTION"
90 PRINTTAB(13)"■6. CHANGE ENTRY"
100 PRINT TAB(13)"■7. QUIT PROGRAM":
    PRINT TAB(13)"■■π■□□
    ENTER CHOICE ?□"
110 GET K$:IF VAL(K$)<1 OR
    VAL(K$)>7 THEN 110
120 C$="":KK$=K$:IF K$="1" THEN
    GOTO 500
130 IF K$="2" THEN GOSUB 440:
    GOTO 640
140 IF K$="3" THEN GOSUB 830:
    GOTO 790
150 IF K$="4" THEN GOSUB 830:
    GOTO 810
160 IF K$="5" THEN PRINT "♥":
    GOSUB 600
170 IF K$="6" AND CO <> 0 THEN
    C$="Y":CQ=1:QQ$=D$(4,1):
    GOTO200
180 IF K$="7" THEN PRINT TAB(13);:
    INPUT"♥□ARE YOU SURE□□□
    ▮▮▮▮"; K$:IFK$="Y"THENEND
190 GOTO 60
200 CC=0:C1=0
210 PRINT"♥■■":IFC$="Y"THEN
    PRINTTAB(12)"■ ENTRY
    NUMBER"CQ
220 PRINT "■π"TAB(20−(LEN(A$
    (VAL(QQ$)))*.5))A$(VAL(QQ$))
230 PRINT "■†-------------
    -------------------------"
240 PRINT "□□πDATE†□□
    □▯□□□□□□□□□πITEM†
    □□□□□□□□▯□□□π
    AMOUNT†"
250 PRINT "----------------
    ----------------------": SC=0
260 IFC$="Y"THENC1=CQ:GOSUB370:
    GOTO860
270 C1=C1+1:IF D$(4,C1)=QQ$ THEN
    GOSUB 370:IF PR$="N" THEN
    SC=SC+1
```

```
Type RETURN in the DATE field for MENU
DATE      ITEM         AMOUNT    CATGY
23/3/84   AIR FARE     £385.00   HOL_
```



```
M A I N   M E N U

1: - MAKE AN ENTRY
2: - VIEW ENTRIES
3: - SAVE TO TAPE
4: - LOAD FROM TAPE
5: - PRINTER OPTION
6: - CHANGE ENTRY
7: - QUIT PROGRAM

- SELECT OPTION. -
```

The main menu (this is the Spectrum version) gives you all these options

```
280 IF SC= >10 THEN SC=Ø:GOSUB
   840:PRINT "▤▣▣▣▣▣"
290 IF C1 =4ØØ OR VAL(D$(4,C1))=Ø
   THEN 31Ø
3ØØ GOTO 26Ø
31Ø C1 =Ø:FOR Z=1 TO 8:N(Z)=Ø:
   NEXT:FR=Ø
32Ø C1 =C1 +1:IF C1 = >4ØØ OR VAL(D$
   (4,C1))=Ø THEN RETURN
33Ø IF VAL(D$(4,C1))=VAL(QQ$)
   THEN N(VAL(QQ$))=N(VAL(QQ$))+
   VAL(D$(3,C1))
34Ø IF VAL(D$(4,C1))=8 AND VAL
   (QQ$)=8 THEN32Ø
35Ø IF VAL(D$(4,C1))< >8 THEN FR=
   FR+VAL(D$(3,C1))
36Ø GOTO 32Ø
37Ø PRINT LEFT$(D$(1,C1)+
   "□□□□□□□□□□□□",9)"◫";
38Ø PRINT LEFT$(D$(2,C1)+
   "□□□□□□□□□□□□□□□□
   □□□□□□",18);"◫£";
39Ø VV$=D$(3,C1):TA=9
4ØØ VV=VAL(VV$):IF VV−INT(VV)=Ø
   THEN VV$=STR$(VV)+".ØØ"
41Ø IF MID$(VV$,LEN(VV$)−1,1)=
   "." THEN VV$=VV$+"Ø"
42Ø PRINT RIGHT$("□□□□□□□□
   □□□□□□□□□□□□"+VV$,TA)
43Ø RETURN
44Ø PRINT "♡π▣▣▣▣▣▣"
   TAB(13)"◪□□□□□
   CATEGORY□□□□▣✝":
   POKE 198,Ø
45Ø FOR Z=1 TO 8:PRINT TAB(12)Z
   ":□"A$(Z):NEXT
46Ø PRINT TAB(13)"▣π◪□□ENTER
   CHOICE□?□"
47Ø GET K$:IF (VAL(K$)<1 OR VAL
   (K$)>8) AND K$< >CHR$(13) THEN
   47Ø
48Ø IF K$=CHR$(13) AND KK$="1"
   THEN 47Ø
49Ø PRINT "♡":QQ$=K$:RETURN
5ØØ IF K$=CHR$(13) THEN
   CO=CO−1: GOTO 6Ø
51Ø CO=CO+1:IF CO>4ØØ THEN
   CO=4ØØ: GOTO 6Ø
52Ø C1 =CO:D$(1,C1)=""
53Ø PRINT "♡▣▣":IF C$< >"Y" THEN
   PRINT "PRESS RETURN IN THE
   DATE FIELD FOR MENU"
54Ø INPUT "▣▣▣π□□ENTER
   DATE✝"; D$(1,C1):IF D$(1,C1)=""
   THEN K$=CHR$(13):GOTO 5ØØ
55Ø INPUT "▣π□□ENTER ITEM✝";
   D$(2,C1)
56Ø INPUT "▣πENTER AMOUNT✝";
   D$(3,C1):GOSUB 44Ø:IF QQ$< >
   CHR$(13)THEN D$(4,C1)=QQ$
57Ø IF QQ$=CHR$(13) THEN 6Ø
58Ø IF C$="Y" THEN QQ$=D$(4,CQ):
   GOTO 2ØØ
59Ø GOTO5ØØ
6ØØ PRINT "▤"TAB(5)"π▣▣▣▣DO
   YOU WANT TO PRINTOUT? (Y/N)"
61Ø GET K$:IF K$="Y" THEN PR$="Y":
   RETURN
62Ø IF K$="N" THEN PR$="N":RETURN
63Ø GOTO 6ØØ
64Ø IF K$=CHR$(13) THEN 6Ø
65Ø IF PR$="Y"THEN OPEN 4,4:CMD 4
66Ø GOSUB 2ØØ:PRINT "π-----------
   ---------------------------✝"
67Ø VV$=STR$(N(VAL(QQ$)))
68Ø PRINT TAB(19)"TOTAL□:π£";:
   TA=12:GOSUB 4ØØ
69Ø PRINT "✝--------------
   -------------------------":
   IF QQ$< >"8" THEN 72Ø
7ØØ PRINT TAB(7)"▣✝TOTAL
   EXPENDITURE□:π£";:TA=12
71Ø VV$=STR$(FR):GOSUB4ØØ:VV$=
   STR$(N(VAL(QQ$))−FR)
72Ø IF QQ$="8" THEN PRINTTAB(17)
   "✝▣BALANCE□:π£";:TA=12:
   GOSUB 4ØØ
73Ø IF PR$="Y" THEN PRINT#4,
   CHR$(13):CLOSE4
74Ø GOSUB94Ø:K$="2":POKE 198,Ø
75Ø GETW$:IF W$=""THEN75Ø
76Ø IF W$=CHR$(13) THEN 6Ø
77Ø IF PR$="Y" THEN GOSUB 44Ø:
   GOTO 64Ø
78Ø GOTO12Ø
79Ø OPEN1,1,1,NM$:PRINT#1,CO:FOR
   Z=1 TO CO:FOR ZZ=1 TO 4:
   PRINT#1,D$(ZZ,Z)
8ØØ NEXT ZZ,Z:CLOSE1:GOTO 6Ø
81Ø OPEN1,1,Ø,NM$:INPUT#1,CO:FOR
   Z=1 TO CO:FOR ZZ=1 TO 4:
   INPUT#1,D$(ZZ,Z)
82Ø NEXT ZZ,Z:CLOSE1:GOTO 6Ø
83Ø INPUT "♡▣πENTER FILE
   NAME✝";NM$:PRINT"♡":RETURN
84Ø PRINT TAB(11)"▣▣(πHIT KEY
   TO CONTINUE)✝":POKE198,Ø:
   WAIT198,1:PRINT"▤▣▣▣▣▣▣";
85Ø FOR Z=1 TO 14:PRINT "□□□□
   □□□□□□□□□□□□□□□□□
   □□□□□□□□□□□□□□□□□
   □□□□□□":NEXT:RETURN
86Ø PRINT"π▣(,) TO MOVE BACK (.)
   TO MOVE FORWARD
   □□□□□□□□□□□□▣SPACE
   BAR TO CHANGE"
87Ø GET P$:IF P$="" THEN 87Ø
88Ø IF P$=CHR$(13) THEN 6Ø
89Ø IF P$="□" THEN 53Ø
9ØØ IF P$="." THEN CQ=CQ+1:
   IFCQ>COTHENCQ=CO
91Ø IF P$="," THEN CQ=CQ−1:
   IFCQ<1THENCQ=1
92Ø IF P$=","ORP$="."THEN QQ$=
   D$(4,CQ):GOTO2ØØ
93Ø GOTO 87Ø
94Ø PRINT"π▣HIT ANY KEY TO
   CONTINUE□□□□□□□□□□
   VIEWING□□▣ PRESS RETURN FOR
   MAIN MENU"
95Ø RETURN
```

DEADLY ENEMIES AND ALIENS

From Space Invaders to the latest arcade games, opponents that shoot back always add to the challenge. Here's how to create them and put them into a complete game routine

Games look much better if you use some of the high resolution graphics features of your machine, rather than just the 'block' characters you've met in this course up till now. Programs using high resolution graphics will be more complex than ones using only keyboard characters, but the results are certainly worth the extra trouble.

Many arcade-type games rely on enemies, aliens or opponents who fire back instead of sitting placidly by while you annihilate them. So this time you'll see a game calked Space Station, suitable for all except the ZX81 and Vic 20, which will teach you how to program lines to move an 'alien' round the screen randomly, and also fire missiles at a target—the space station.

The player has four shields which he can use to ward off the marauding alien's missiles. You can't keep the shields up all the time, though, because there's only a limited amount of fuel to power the shields.

To make the game more difficult, not only is the program designed to move the alien randomly, but the alien may be made to disappear into hyperspace and reappear somewhere entirely different.

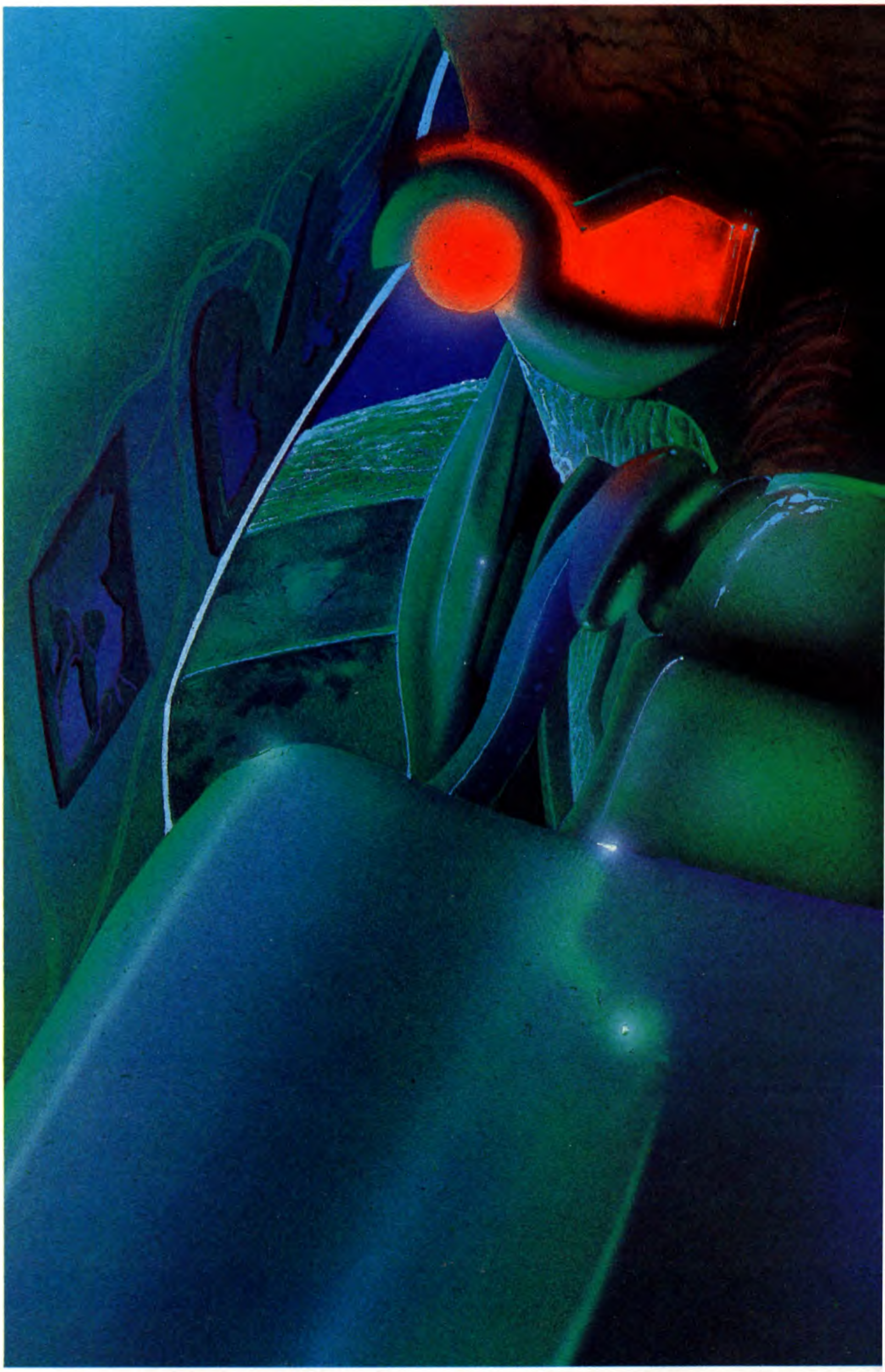
As it stands, the game isn't really complete—no timing or scoring has been added. But this is easily remedied using the methods given on pages 97 to 1Ø3.

Although the alien is like those in commercial games, the space station is only an outline. If you want to spend an interesting half hour, you can redesign it using user defined graphics as described on page 38—or, for Commodore 64 owners, sprites as described on page 151. You should, however, keep within the area used by our space station. Otherwise it could overlap the defensive shields, necessitating much revision.

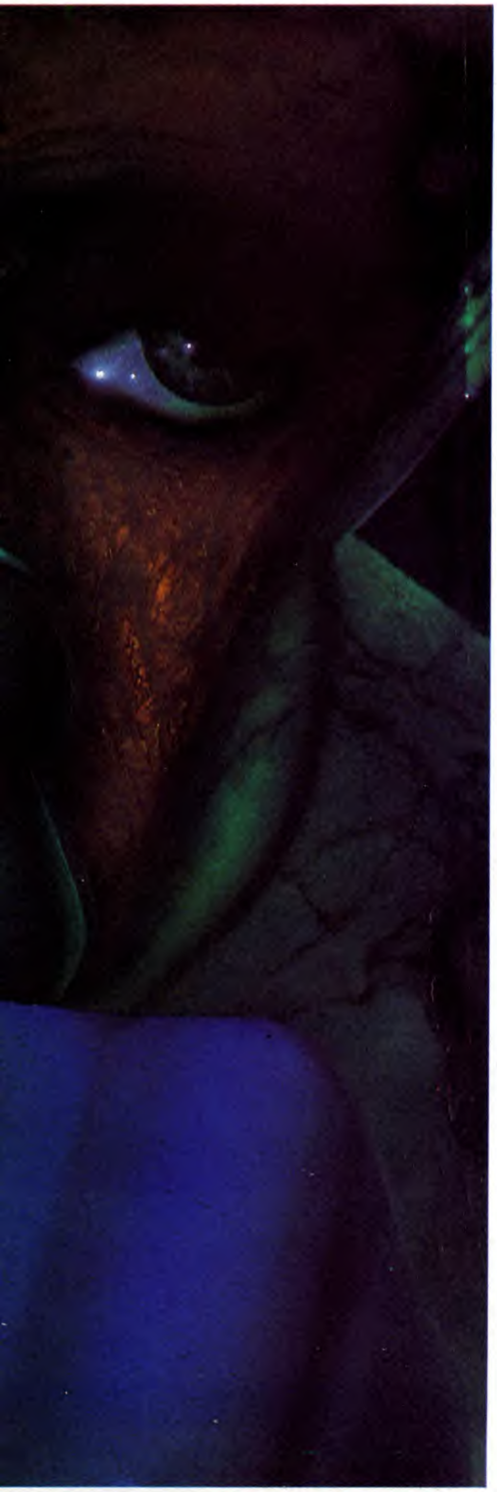
T

Type in and RUN the first part of the game:

```
1Ø PCLEAR4:PMODE4,1:PCLS
15 SCREEN 1,1
2Ø DIM AL(6),BL(6),BO(4)
3Ø DEFFNZ(X)=SGN(X)*SQR(V*V*X*X/
   ((127−AX)*(127−AX)+(95−AY)*
   (95−AY)))
```



- THE IDEA OF THE ALIEN GAME ROUTINE
- HOW TO DRAW THE GAME ELEMENTS ON THE SCREEN
- ADDING MOVEMENT
- SHIELDS YOUR—PROTECTION
- MISSILES—THE ELEMENT OF DANGER
- PLAYING THE GAME
- CREATING AN ENDING



```
4Ø LET PW=25Ø
5Ø FORI=ØTO7:READA:POKE1536+
   I*32,A:NEXTI
6Ø GET (Ø,Ø)—(7,7),AL,G
65 GOTO 65
25Ø DATA 24,126,9Ø,126,126,195,
   129, 129
```

You'll see the alien appear on the screen.

The high resolution graphics are set up by Line 1Ø. At this stage the screen is switched on by Line 15 so that you can see how the program works, but the line will be removed later on in the development of the complete alien game.

The arrays into which the alien, the missile and a blank will be fed are DIMensioned by Line 2Ø. Line 3Ø uses a keyword that you haven't seen yet. DEFFN is short for 'define function'. If you have a long mathematical expression it saves you having to type it out in full several times in a program. The mathematical expression in Line 3Ø is now called FNZ and is used later on in the program to move the missile diagonally across the screen. Line 4Ø sets the end of the fuel gauge.

Line 5Ø draws the alien on the screen by READing the DATA in Line 25Ø and POKEing it on to the top left of the screen. The alien is 'remembered' by the Dragon by the GET in Line 6Ø. The alien is now in array AL.

Line 65 is also a temporary line. All it does is keep the screen switched on. And once again it will be removed later on, when the rest of the game is added to the program.

DRAWING THE MISSILE

Next add these lines and RUN the program.

```
7Ø FOR J=Ø TO 4:READ A:POKE
   1536+J*32,A:NEXTJ
8Ø GET(Ø,Ø)—(4,4),BO,G
85 GOTO 85
26Ø DATA 32,112,248,112,32
```

The missile that will be fired by the alien is POKEd into the top left of the screen. It doesn't matter that the missile is being POKEd on top of the alien, nor that there are still some bits of alien left, because Line 8Ø only GETs the area occupied by the missile and not the surroundings.

DRAWING THE SPACE STATION

Remove Line 85 by typing 85 ENTER and add these lines. Then RUN the program.

```
9Ø LET AX=RND(248)—1: LET AY=
   RND(178)+5
1ØØ PCLS
11Ø CIRCLE(127,95),12,5:DRAW
   "BM127,95;C5S48NUNLNDNR"
115 GOTO 115
```

A random start position for the alien is chosen by Line 9Ø. Line 1ØØ clears the screen before Line 11Ø draws the space station. The DRAW command at the end of the line draws a cross on the space station. DRAW will have to wait until later for a full explanation, but can be thought of as a succession of LINE statements, as already explained in BASIC Programming.

DRAWING THE FUEL GAUGE

Remove Line 115 in the same way as you removed Lines 65 and 85. Add these lines and more graphic detail will appear on the screen:

```
12Ø DRAW"BM131,87;S4D5BD6BLR3D2L3
   D2R3BL12R3U2NL3U2NL3BU6U5G4R3"
13Ø DRAW"BM5,1;L4D2NR4D2BE4BR2D4
   R3U4BR5L3D2NR3D2R3BE4D4R3"
14Ø LINE(25,1)—(PW,5),PSET,BF
145 GOTO 145
```

Line 12Ø draws numbers on the space station which correspond to the shield numbers. Line 13Ø displays the word FUEL. Unfortunately, the Dragon cannot display ordinary keyboard characters on the high resolution screen, so letters or numbers must be DRAWn.

The full fuel gauge for the shields is displayed by Line 14Ø. The line uses a quick method for drawing rectangles. You use LINE to draw a line from the top left corner to the bottom right corner. PSET tells the Dragon to draw the line in buff in this mode and colour set. BF is short for 'box fill' and fills the rectangle with the colour used for the original line. If you want to draw an empty rectangle use B instead of BF.

The high resolution graphics for the game are now complete. The rest of the program is concerned with moving the alien and the missile and activating the shields.

MOVING THE ALIEN

There are three subroutines to add to the program. This one is concerned with moving the alien. Type it in but don't RUN it because nothing will happen yet.

```
1ØØØ LETLX=AX:LETLY=AY
1Ø1Ø IF RND(1Ø)=1 THEN LETAX=RND
    (248)-1:LETAY=RND(178)+5
1Ø2Ø LETAX=AX+RND(15)-8:LETAY=
    AY+ RND(15)-8
1Ø3Ø IF AX>1Ø3 AND AX<144 AND
    AY>71 AND AY<112 THEN LETAX=LX:
    LETAY=LY
1Ø4Ø IFAX<Ø THEN LETAX=-AX
1Ø5Ø IFAX>248 THEN LETAX=497-AX
1Ø6Ø IFAY<6 THEN LETAY=12-AY
1Ø7Ø IFAY>184 THEN LETAY=369-AY
1Ø8Ø PUT(LX,LY)-(LX+7,LY+7),
    BL,PSET
1Ø9Ø PUT(AX,AY)-(AX+7,AY+7),
    AL,PSET
11ØØ RETURN
```

The alien is controlled somewhat like the missile and missile base movement covered on pages 54 to 59. Line 1ØØØ sets the last position coordinates equal to the current position coordinates before the alien is moved.

So that the alien may suddenly shift around the screen, Line 1Ø1Ø chooses a random number. If the random number is 1, then the alien jumps to a new screen position. If the random number isn't 1 a new position for the alien is chosen between −7 and +7 pixels away in the x direction (left to right) and the same range of distances in the y direction (up and down)—see Line 1Ø2Ø.

Line 1Ø3Ø stops the alien overrunning the space station, while Lines 1Ø4Ø to 1Ø7Ø stop the alien being displayed off the screen.

The alien is blanked out in line 1Ø8Ø by PUTing a series of blank graphics over its last position, and the alien is PUT into its new position by Line 1Ø9Ø.

Line 11ØØ RETURNs the program to Line 16Ø—which, incidentally, you haven't yet entered.

FIRING THE MISSILE

The next subroutine first decides whether to fire a missile, then sets up the missile's position on the screen and finally checks which shield will block the missile.

```
2ØØØ IFRND(7)<>1 THENRETURN
2Ø1Ø V=RND(8)+5:DX=FNZ(127-AX):
    DY=FNZ(95-AY)
2Ø2Ø IF DX<=Ø AND DY>=Ø THEN
    LETMA=1:GOTO 2Ø5Ø
2Ø3Ø IF DX<=Ø AND DY<=Ø THEN
    LETMA=2:GOTO 2Ø5Ø
2Ø4Ø IF DX>=Ø AND DY<=Ø THEN
    LETMA=3 ELSE LETMA=4
2Ø5Ø LETMX=AX:LETMY=AY
2Ø6Ø PUT(MX,MY)-(MX+4,MY+4),
    BO,OR
2Ø7Ø LETAF=1:RETURN
```

Line 2ØØØ decides whether to fire a missile. There's a six-to-one chance that it will, but the program can't fire if a missile is already on the screen. If a missile isn't to be fired then the subroutine ends.

Line 2Ø1Ø sets how large the steps taken by the missile will be—you could, perhaps, think of V as velocity, or speed. V is plugged into FNZ—defined in Line 3Ø—in the last part of the line so that the new position of the missile can be worked out.

Lines 2Ø2Ø to 2Ø4Ø find out which quarter of the screen the missile is in and also which shield is needed to block the missile—MA is the missile angle, if you like.

Line 2Ø5Ø starts the missile's flight from the alien's position. And Line 2Ø6Ø PUTs the missile on the screen before Line 2Ø7Ø makes AF=1, which tells the Dragon that a missile has been fired. The subroutine ends.

The final subroutine is from Lines 3ØØØ to 3Ø7Ø. Type in the lines, but again RUNning will have no effect.

```
3ØØØ PUT(MX,MY)-(MX+4,MY+4),
    BL,PSET
3Ø1Ø LETMX=MX+DX:LETMY=MY+DY
3Ø2Ø IFMX>11ØANDMX<14ØANDMY
    >79AND MY<1Ø8 THEN GOTO 3Ø5Ø
3Ø3Ø PUT(MX,MY)-(MX+4,MY+4),BO,OR
3Ø4Ø RETURN
3Ø5Ø IF SH(MA)=Ø THEN GOTO 3Ø7Ø
3Ø6Ø LETAF=Ø:RETURN
3Ø7Ø CLS:PRINT@256,"BANG..YOUR
    SHIELDS WERE DOWN !"
```

The missile is blanked out by Line 3ØØØ. The missile's new position is worked out by Line 3Ø1Ø. Line 3Ø2Ø checks if the missile has reached the shields. If it has then the program jumps to Line 3Ø5Ø where there is a check to see if the right shield is up. If there is no blocking shield the program finishes with the message BANG ... YOUR SHIELDS WERE DOWN!

If the missile hasn't yet reached the shields, Line 3Ø5Ø PUTs it on the screen at the new position. Line 3Ø4Ø ends the subroutine.

This completes the program:

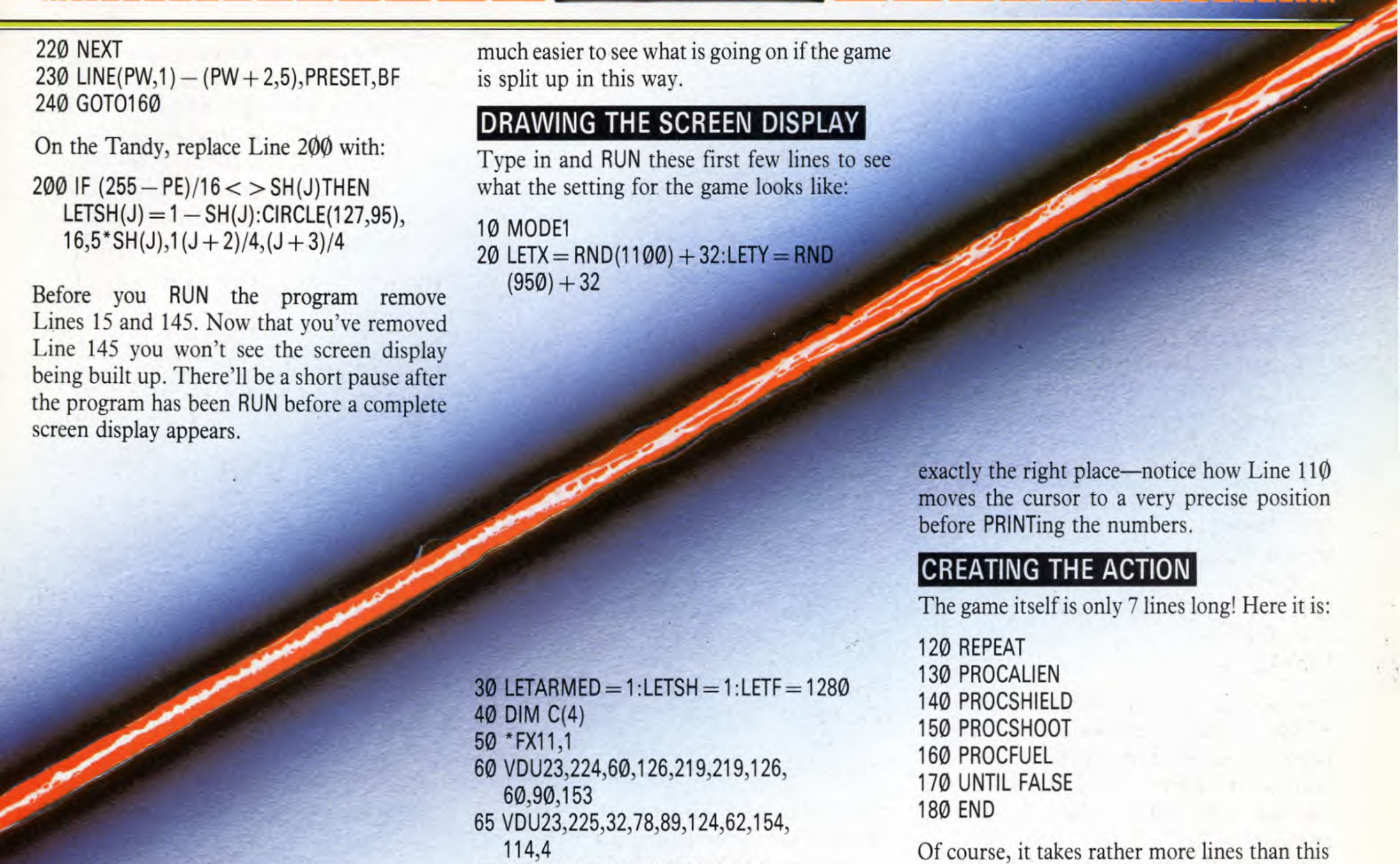
```
15Ø SCREEN 1,1
16Ø IFAF=Ø THEN GOSUB 2ØØØ
    ELSEGOSUB3ØØØ
17Ø GOSUB1ØØØ
18Ø FOR J=1 TO 4
19Ø LETPE=PEEK(338+J):IF PW<25
    THEN LETPE=255
2ØØ IF 255-PE <> SH(J) THEN
    LETSH(J)=1-SH(J):CIRCLE(127,95),
    16,5*SH(J),1,(J+2)/4,(J+3)/4
21Ø IFSH(J)=1 THEN LETPW=PW-2
```

```
22Ø NEXT
23Ø LINE(PW,1)-(PW+2,5),PRESET,BF
24Ø GOTO16Ø
```

On the Tandy, replace Line 2ØØ with:

```
2ØØ IF (255-PE)/16<>SH(J)THEN
    LETSH(J)=1-SH(J):CIRCLE(127,95),
    16,5*SH(J),1(J+2)/4,(J+3)/4
```

Before you RUN the program remove Lines 15 and 145. Now that you've removed Line 145 you won't see the screen display being built up. There'll be a short pause after the program has been RUN before a complete screen display appears.



The screen is now switched on by Line 15Ø. Line 16Ø checks if a missile has been fired. If AF=Ø no missile has been fired, and the program jumps to the subroutine concerned with firing a missile—the one starting at Line 2ØØØ—or else the program jumps to the subroutine which moves the missile—the one starting at Line 3ØØØ. Next, the alien is moved. Line 17Ø makes the program jump to the subroutine starting at Line 1ØØØ.

The section of program from Lines 18Ø to 22Ø is concerned with activating the shields. Line 19Ø checks which key is being pressed. If the key is a number from 1 to 4 Line 2ØØØ will draw the shield, and if any of the shields are activated Line 21Ø subtracts from the fuel.

Line 23Ø draws a black rectangle at the end of the fuel display, giving the impression that the fuel supply is going down as PW decreases.

Finally, Line 24Ø starts the process again.

On the Acorn machines the space station game is divided into two main sections. The first part draws the screen display and does all the 'fiddly bits' like setting the initial value of the variables and defining the UDG characters. The second part of the program deals with the action part of the game, such as moving the alien and firing the missile. It is much easier to see what is going on if the game is split up in this way.

DRAWING THE SCREEN DISPLAY

Type in and RUN these first few lines to see what the setting for the game looks like:

```
1Ø MODE1
2Ø LETX=RND(11ØØ)+32:LETY=RND
   (95Ø)+32
3Ø LETARMED=1:LETSH=1:LETF=128Ø
4Ø DIM C(4)
5Ø *FX11,1
6Ø VDU23,224,6Ø,126,219,219,126,
   6Ø,9Ø,153
65 VDU23,225,32,78,89,124,62,154,
   114,4
7Ø MOVEØ,1ØØØ:MOVE128Ø,1ØØØ:
   PLOT85,Ø,1Ø24:PLOT85,128Ø,1Ø24
8Ø PRINT "FUEL":VDU5
9Ø MOVE56Ø,512:DRAW64Ø,592:
   DRAW72Ø,512:DRAW64Ø,432:
   DRAW56Ø,512
1ØØ MOVE64Ø,592:DRAW64Ø,432:
    MOVE56Ø,512:DRAW72Ø,512
11Ø MOVE592,552:PRINT "4□1":
    MOVE592,5Ø4:PRINT "3□2"
```

The first few lines mostly just set the variables. X and Y are the start position of the alien; ARMED=1 means the alien starts with a missile (ARMED=Ø means no missile); SH=1 means that the shields are operational (again, Ø means that they are not); and F is the fuel level. DIM C(4) dimensions the array for the colour of the four shields and *FX11,1 speeds up the auto-repeat on the keys—essential for any game that relies on keyboard input.

Lines 6Ø and 65 define the UDG characters for the alien and the missile, although they are not actually PRINTed out until later in the program.

The next lines draw the display. Lines 7Ø and 8Ø draw the fuel gauge and Lines 9Ø to 11Ø draw the space station and number its sectors 1 to 4. The VDU 5 in Line 8Ø causes text to be PRINTed at the graphics cursor. This makes it easier to PRINT the sector numbers in exactly the right place—notice how Line 11Ø moves the cursor to a very precise position before PRINTing the numbers.

CREATING THE ACTION

The game itself is only 7 lines long! Here it is:

```
12Ø REPEAT
13Ø PROCALIEN
14Ø PROCSHIELD
15Ø PROCSHOOT
16Ø PROCFUEL
17Ø UNTIL FALSE
18Ø END
```

Of course, it takes rather more lines than this to define each procedure, but you can see that the structure of the game is very simple indeed.

Lines 1ØØ and 15Ø just keep looping through the list of procedures (UNTIL FALSE means 'carry on for ever'). PROCALIEN moves the alien, PROCSHIELD activates the shields around the space station, PROCSHOOT releases the missiles and detects if you are protected by the correct shield, and PROCFUEL measures how much fuel you have left.

If you try to RUN the program now you will get a 'No such FN/PROC' error as none of the procedures have been defined. That is the next task.

MOVING THE ALIEN

Here are the lines to move the alien:

```
19Ø DEF PROCALIEN
2ØØ GCOLØ,Ø:MOVEX,Y:VDU224
21Ø IF RND(2ØØ)=123 THEN X=64Ø:
    Y=512:GOTO 28Ø
22Ø LETDX=RND(4Ø)-2Ø:LETDY=RND
    (4Ø)-2Ø
23Ø IF X>12ØØ THEN DX=-ABS(DX)
24Ø IF X<1Ø THEN DX=ABS(DX)
25Ø IF Y<5Ø THEN DY=ABS(DY)
26Ø IF Y>95Ø THEN DY=-ABS(DY)
27Ø LETX=X+DX: LET Y=Y+DY
```

```
280 IF X>5ØØ AND X<75Ø AND Y>4ØØ
    ANDY<65Ø THEN X=RND(125Ø):
    Y=RND (99Ø)+3Ø:GOTO28Ø
29Ø GCOLØ,2:MOVEX,Y:VDU224
3ØØ ENDPROC
```

Try RUNning the program now. It will work if you put in a temporary extra line:

```
135 GOTO17Ø
```

But remember to delete this line once you have seen what the procedure does. Delete it by typing 135 followed by RETURN.

The alien moves around randomly, taking very short steps most of the time but occasionally jumping to a different part of the screen.

Line 22Ø works out the small random steps DX and DY, and Line 21Ø controls the big jumps across the screen. It only jumps if RND(2ØØ) equals 123. The number 123 is a dummy number and it could, in fact, be any number between 1 and 2ØØ; it just means there is a 1 in 2ØØ chance of it turning up. You can change the numbers to something smaller if you want the alien to jump about more often—RND(1ØØ)=5Ø for example.

Lines 23Ø to 26Ø check if the alien is nearing the edge of the screen and if it is they make sure that DX and DY are either added on or taken away to bring it back towards the centre.

Line 27Ø adds on the small step, Line 28Ø makes it jump to a new position if it gets too near the space station, and Line 29Ø chooses yellow graphics and plots the alien. (VDU224 means PRINT CHR$224 which is the code number given to the alien UDG.)

The only line that hasn't been described so far is Line 2ØØ. This simply blanks out the previous position of the alien.

ACTIVATING THE SHIELDS

PROCSHIELD is the next procedure on the list:

```
31Ø DEF PROCSHIELD
32Ø LETC(1)=Ø:LETC(2)=Ø:LETC(3)=Ø:
    LETC(4)=Ø
33Ø IF SH=Ø THEN GOTO 42Ø
34Ø LETA$=INKEY$(1)
35Ø *FX15,1
36Ø IF A$=“” THEN GOTO 42Ø
37Ø IF A$=“1” THEN C(1)=3
38Ø IF A$=“2” THEN C(2)=3
39Ø IF A$=“3” THEN C(3)=3
4ØØ IF A$=“4” THEN C(4)=3
41Ø LETF=F−2
42Ø GCOL Ø,C(1):MOVE64Ø,6Ø8:DRAW736,
    512:GCOL Ø,C(2):DRAW64Ø,416:GCOL
    Ø,C(3):DRAW544,512:GCOL Ø,C(4):
    DRAW64Ø,6Ø8
43Ø ENDPROC
```

This displays the shield which you use to protect your space station from the missiles. You can protect only one sector at a time and you activate the shields by pressing one of the keys 1 to 4.

Line 32Ø starts by setting the colour of each shield to Ø—ie, black (an activated shield is white). Line 33Ø checks if your shields are operational—if SH=Ø it means your fuel has run out and you cannot use the shields.

The next few lines detect the keypress and turn the colour of the relevant shield to colour 3 (ie, white). Line 41Ø reduces your fuel by 2 units each time a shield is used—so turn them on only when you really need to!

Finally, Line 42Ø draws the four shields. It draws all four but you see only the white one; the other three are still black and therefore invisible. This is a simple way of displaying the correct shield without a lot of extra and complicated program lines. But note that if SH in Line 33Ø were Ø then all four shields would be black.

FIRING THE MISSILES

PROCSHOOT gives the alien a 1 in 3Ø chance of firing a missile at the space station and it also checks whether a missile gets through your defences. Enter these lines now:

```
44Ø DEF PROCSHOOT
45Ø IF ARMED=Ø THEN GOTO 54Ø
46Ø IF RND(3Ø)<>13 THEN GOTO 59Ø
47Ø LETARMED=Ø:LETFX=(624−X)/15:
    LETFY=(528−Y)/15
48Ø LETG=X+FX:LETH=Y+FY
49Ø GCOL3,2:MOVEG,H:VDU225
5ØØ IF G>624 AND H>528 THEN S=1
51Ø IF G>624 AND H<528 THEN S=2
52Ø IF G<624 AND H<528 THEN S=3
53Ø IF G<624 AND H>528 THEN S=4
54Ø GCOL3,2:MOVEG,H:VDU225
55Ø LETG=G+FX:LETH=H+FY
56Ø IF NOT(G>584 AND G<684 AND
    H>48Ø AND H<58Ø) THEN MOVE
    G,H: VDU225:ENDPROC
57Ø LETARMED=1
58Ø IF C(S)=Ø THEN PROCFINISH
59Ø ENDPROC
```

Line 45Ø checks if the alien is armed. If

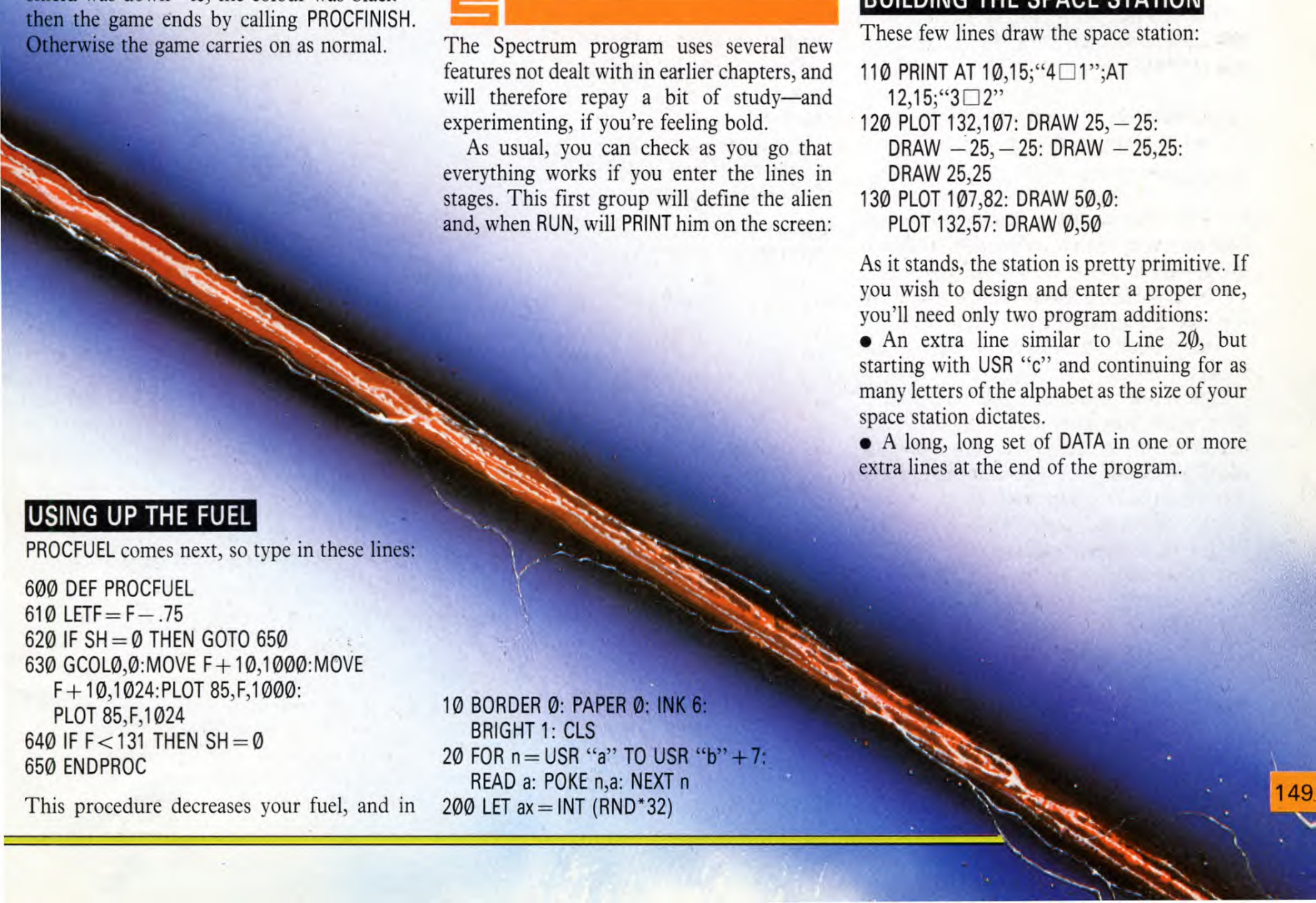
ARMED equals Ø it means a missile has already been released so this line jumps to a later part of the program. Line 46Ø jumps to the end of the procedure unless RND(3Ø) = 13. The 13 is a dummy number again like the 123 in Line 21Ø. It gives the alien a 1 in 3Ø chance of firing a missile.

Assuming the conditions in Line 45Ø and 46Ø are not true then the computer eventually gets to Line 47Ø. This immediately fires a missile and then calculates FX and FY which is how far the missile moves. These are worked out so that the missile always heads towards the space station. Line 48Ø adds this to the position of the alien to give the position of the missile—G and H are the missile's coordinates. Then Line 49Ø plots a yellow missile.

The next four lines work out which sector the missile is in and they set the variable S to the number of the sector.

Line 54Ø blanks out the old position of the missile, Line 55Ø adds on another step and Line 56Ø PRINTs it at the new position if it is *not* near to the space station.

The program gets to Line 57Ø only when the missile gets to the base. The alien is immediately armed again (ARMED = 1), and then it checks the colour of the shield. If the shield was down—ie, the colour was black—then the game ends by calling PROCFINISH. Otherwise the game carries on as normal.



USING UP THE FUEL

PROCFUEL comes next, so type in these lines:

```
600 DEF PROCFUEL
610 LETF=F−.75
620 IF SH=Ø THEN GOTO 650
63Ø GCOLØ,Ø:MOVE F+1Ø,1ØØØ:MOVE
   F+1Ø,1Ø24:PLOT 85,F,1ØØØ:
   PLOT 85,F,1Ø24
64Ø IF F<131 THEN SH=Ø
65Ø ENDPROC
```

This procedure decreases your fuel, and in Line 63Ø rubs out the fuel gauge to show how little you have left. Remember your fuel decreases more quickly when your shields are up—look at Line 41Ø again.

When the fuel level drops below 131, Line 64Ø turns off your shields by setting SH to Ø.

ENDING THE GAME

If a bomb gets through your defences then you've lost the game. This is done with PROCFINISH:

```
66Ø DEF PROCFINISH
665 FOR D=1 TO 2ØØØ: NEXT
67Ø VDU4:CLS:PRINT'"BANG! YOUR
   SHIELDS WERE DOWN"
68Ø *FX12,Ø
685 FOR D=1 TO 2ØØØ: NEXT
69Ø *FX15,1
7ØØ END
```

All it does is to tell you you're dead and reset various things that were set earlier. The VDU 4 resets the VDU 5 statement. *FX12,Ø resets the auto-repeat on the keys to normal, and *FX15,1 empties the keyboard buffer that may be filled with lots of 1s, 2s, 3s and 4s that you pressed for the shields. In short, it 'tidies things up' before finally ending the game.

The Spectrum program uses several new features not dealt with in earlier chapters, and will therefore repay a bit of study—and experimenting, if you're feeling bold.

As usual, you can check as you go that everything works if you enter the lines in stages. This first group will define the alien and, when RUN, will PRINT him on the screen:

```
1Ø BORDER Ø: PAPER Ø: INK 6:
   BRIGHT 1: CLS
2Ø FOR n=USR "a" TO USR "b"+7:
   READ a: POKE n,a: NEXT n
2ØØ LET ax=INT (RND*32)
21Ø LET ay=INT (RND*21)+1
22Ø IF ax>11 AND ax<21 AND ay>6
   AND ay<16 THEN GOTO 2ØØ
49Ø PRINT INK 4;AT ay,ax;CHR$ 144
8ØØ DATA 6Ø,126,219,219,126,6Ø,9Ø,153,
   Ø,Ø,24,6Ø,6Ø,24,Ø,Ø
```

Lines 2Ø and 8ØØ define the alien and his missile (which is not yet visible). They use the technique that was explained fully in Machine Code 2. (FOR n = USR "a" TO USR "b" + 7 . . . POKE n,a means the same thing as FOR n = Ø to 15 . . . POKE USR "a" + n,a.)

Lines 2ØØ and 21Ø start the alien off at a random position on the screen, and Line 49Ø PRINTs him. (The PRINT CHR$ 144 in this line means the same as the PRINT < graphics "a" > in the earlier article.)

Line 22Ø looks odd at this stage but, as you'll see as the program progresses, is the means of preventing the alien popping up in the middle of your space station.

For now, you may feel like omitting Line 1Ø, because having your program listing in yellow on a black screen makes it harder to read. If you do this, you must remember to reinstate it later, or Line 64Ø (explained below) will not work.

BUILDING THE SPACE STATION

These few lines draw the space station:

```
11Ø PRINT AT 1Ø,15;"4□1";AT
   12,15;"3□2"
12Ø PLOT 132,1Ø7: DRAW 25,−25:
   DRAW −25,−25: DRAW −25,25:
   DRAW 25,25
13Ø PLOT 1Ø7,82: DRAW 5Ø,Ø:
   PLOT 132,57: DRAW Ø,5Ø
```

As it stands, the station is pretty primitive. If you wish to design and enter a proper one, you'll need only two program additions:

- An extra line similar to Line 2Ø, but starting with USR "c" and continuing for as many letters of the alphabet as the size of your space station dictates.
- A long, long set of DATA in one or more extra lines at the end of the program.

PRINTING THE MISSILE

The next job is to PRINT the alien's missile, and plot its path towards the space station:

```
150 LET mf=Ø
300 IF mf=1 THEN GOTO 4ØØ
31Ø IF RND<.9 THEN GOTO 42Ø
32Ø LET mf=1: LET my=ay: LET
   mx=ax: LET fy=11−my: LET
   fx=16−mx
33Ø LET b=1: IF ABS fy>ABS fx
   THEN LET b=2
34Ø IF b=1 THEN LET sx=SGN fx:
   LET sy=SGN fy*ABS (fy/fx)
35Ø IF b=2 THEN LET sy=SGN fy:
   LET sx=SGN fx*ABS (fx/fy)
4ØØ PRINT AT my,mx;"□": LET
   my=my+sy: LET mx=mx+sx:
   PRINT INK 5;AT my,mx;CHR$ 145: IF
   my>1Ø AND my<12 AND mx>15 AND
   mx<17 THEN GOTO 7ØØ
62Ø IF RND>.9 THEN PRINT AT ay,ax;
   "□": GOTO 2ØØ
63Ø IF mf=Ø THEN GOTO 3ØØ
65Ø GOTO 3ØØ
7ØØ CLS : PRINT FLASH 1; PAPER 2;AT
   1Ø,1;" BANG!□Your shields were
   down□"
```

This whole section, as you can see from Line 65Ø, is a loop that the computer traverses several times when the alien appears.

Line 15Ø sets the whole scene to zero: there is no missile coming at you—yet.

Line 31Ø decides whether the alien will fire a missile at you during this particular loop of the program (there's a 9 to 1 chance he will).

If there is a missile, Line 32Ø sets its starting position (my, mx) at the obvious place—where the alien is (ax, ay). The middle bit of Line 4ØØ PRINTs the missile, using CHR$145 instead of graphics "B".

The piece of program from the latter half of Line 32Ø to Line 4ØØ is the crafty bit. What it does is to take the numbers for the middle of the space station, and the numbers representing the alien's current position, then subtract the latter from the former so that the missile 'homes in' on the space station.

Since some of the numbers involved are negative (for leftwards and downwards travel) and some positive (for rightwards and upwards travel) you may find it difficult to follow this block if you do not understand ABS and SGN, which are covered in a later chapter. But here are some clues:

The second half of Line 32Ø deducts the missile's current position (my, mx) from the centre point of the space station (screen position 11, 16) and calls the resulting coordinates fy and fx.

Lines 33Ø to 35Ø, using ABS and SGN, add 'course correction' factors (sy and sx) to fy and fx. Line 4ØØ starts by unPRINTing the missile at its old position. Then it adds the sx and sy numbers to the old position, ready for the missile to be PRINTed again, one step closer.

MOVING THE ALIEN

Now that the alien has fired its missile it is time for him to move on. So add these lines:

```
42Ø LET xx=ax: LET yy=ay: LET
   m=INT (RND*4)
43Ø IF m=Ø AND ax<31 THEN LET
   xx=ax+1
44Ø IF m=1 AND ax>Ø THEN LET
   xx=ax−1
45Ø IF m=2 AND ay<21 THEN LET
   yy=ay+1
46Ø IF m=3 AND ay>1 THEN LET
   yy=ay−1
47Ø IF xx>11 AND xx<21 AND yy>6
   AND yy<16 THEN GOTO 49Ø
48Ø PRINT AT ay,ax;"□": LET
   ax=xx: LET ay=yy
```

First the Spectrum decides in which direction the alien will move. Once this is done by the random number in Line 42Ø, the purpose of Lines 43Ø to 46Ø becomes obvious—they are conventional movement lines. Line 47Ø keeps the alien out of the station.

Line 4ØØ (entered earlier) records a hit, directing the program to Line 7ØØ if the missile hits the middle of the station. You may wonder why this line uses >1Ø and <12, rather than the simpler 11, and >15 and <17, rather than 16. But remember: although the computer can only PRINT the alien at a whole number, the numbers moving him *are a series of decimalized fractions*. So the chance of their actually becoming 11, 16 are remote.

Finally, after about ten loops, Line 62Ø blots the alien out at its final position on this loop and starts it again at Line 2ØØ.

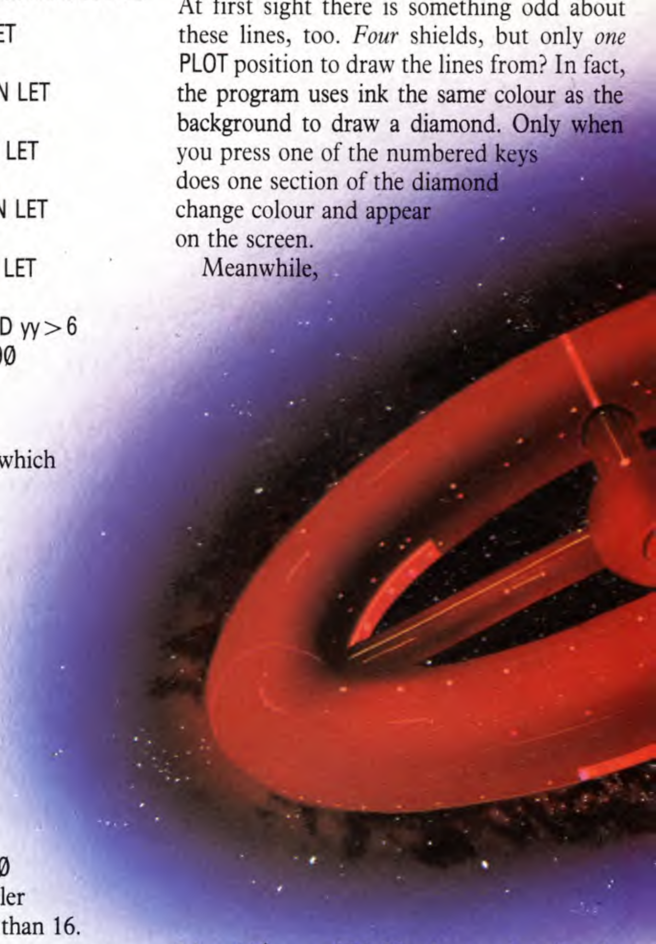
BUILDING THE SHIELDS

These lines build the shields to ward off the approaching missile:

```
14Ø PLOT INVERSE 1;132,122
5ØØ DIM a(4)
51Ø LET a$=INKEY$: IF a$="" THEN
   GOTO 6ØØ
52Ø IF a$="1" THEN LET a(1)=1
53Ø IF a$="2" THEN LET a(2)=1
54Ø IF a$="3" THEN LET a(3)=1
55Ø IF a$="4" THEN LET a(4)=1
6ØØ DRAW INK a(1)*4, INVERSE 1−a(1),
   4Ø,−4Ø: DRAW INK a(2)*4, INVERSE
   1−a(2),−4Ø,−4Ø: DRAW INK a(3)*4,
   INVERSE 1−a(3),−4Ø,4Ø: DRAW INK
   a(4)*4, INVERSE 1−a(4),4Ø,4Ø
64Ø IF ATTR (my,mx)=68 THEN PRINT
   AT my,mx;"□": LET mf=Ø
```

At first sight there is something odd about these lines, too. *Four* shields, but only *one* PLOT position to draw the lines from? In fact, the program uses ink the same colour as the background to draw a diamond. Only when you press one of the numbered keys does one section of the diamond change colour and appear on the screen.

Meanwhile, Line 64Ø uses ATR 68—the number for the colour of the shields—to repel the missile by unPRINTing if it hits the shield.



CLEANING UP

The remaining lines are very easy to follow. They set the fuel supply to 1ØØ and make it dwindle until, in the middle of Line 51Ø (now amended) the shields become inactive. Remember to reinstate Line 1Ø:

```
1ØØ PRINT PAPER 2; INK 6;AT Ø,Ø;
   "□FUEL□"
16Ø LET fu=1ØØ
51Ø LET a$=INKEY$: IF a$="" OR
   fu=Ø THEN GOTO 6ØØ
56Ø LET fu=fu−1
61Ø PRINT PAPER 3; INK 7;AT Ø,6;
   "□";fu;"□"
```

The Commodore 64 version of the space station game uses sprites, information for which is contained within the large number of DATA statements near the beginning of the program, as explained on page 15.

```
10 POKE 56,100:POKE 55,0:POKE 52,
   100:POKE 51,0:CLR
20 DATA0,254,0,3,57,128,7,255,192,0,
   16,0,16,56,16,56,84,56,124,146,
   124,131,255
30 DATA130,144,56,18,184,16,58,144,
   16,18,131,255,130,254,84,254,252,
   56,126,0,56
40 DATA0,0,40,0,0,56,0,1,199,0,6,16,
   192,1,199,0,0,124,0
50 DATA0,0,0,127,255,254,64,0,2,64,
   0,2,64,16,2,64,48,2,64,16,2,64,
   16,2,80,56
60 DATA114,84,0,18,94,0,114,68,0,66,
   68,56,114,64,8,2,64,56,2,64,8,
   2,64
70 DATA56,2,64,0,2,64,0,2,127,255,
   254,0,0,0
80 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,8,0,0,42,0,0,20,
   0,0,42,0,0
90 DATA42,0,0,73,0,0,8,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
100 DATA0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0,0,0,0,0,3,3,0,4,132,
   128,255
110 DATA255,255,72,72,68,48,48,56,0,
   0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
   0,0,0,0,0,0
120 DATA0,0,16,0,0,24,0,0,20,0,0,20,
   0,0,20,0,0,24,0,0,48,0,0,80,0,0,
   80,0,0,48,0
130 DATA0,24,0,0,20,0,0,20,0,0,24,0,
   0,48,0,0,80,0,0,80,0,0,80,0,0,48,
   0,0,16
140 DATA0,0,16,0
210 FOR ZZ=0 TO 4:POKE 2040+ZZ,
   200+ZZ
220 FOR Z=1TO63:READ X:POKE 12799+
   (ZZ*64)+Z,X:NEXT Z,ZZ
230 CLR:V=53248:FU=100:POKE 650,255:
   POKE 53280,0:POKE 53281,0:
   PRINT "♥"
240 POKE V+2,145:POKE V+3,120:
   POKE V+23,250:POKE V+29,250:
   POKE V+30,240
   250 XX=31+INT(RND(1)*210):YY=60:
      DX=1:DY=1:IF RND(1)>.50 THEN
                     YY=180:DY=-DY
    260 PRINT "☰"TAB(14)"πFUEL:□□
                 □□▮▮▮▮▮▮▮▮" FU
270 IFRND(1)>.90THENXX=31+INT(RND
   (1)*210):YY=60:DX=1:DY=1
320 XX=XX+DX: IF XX=<30 OR
   XX=>245 THEN DX=-DX
330 YY=YY+DY:IF YY=<50 OR
   YY=>190 THEN DY=-DY
340 POKE V,XX: POKE V+1,YY:IF F=0
   THEN F=1:FX=XX:FY=YY
350 IF F=1 THEN GOSUB 410
360 GET A$:S$="":SH=0
370 IF A$="4" OR A$="2" THEN SH=1:
   POKE 2043,204:S$=A$
380 IF A$="1" OR A$="3" THEN SH=1:
   POKE 2043,203:S$=A$
390 IF SH=1 THEN 470
400 POKE V+21,247:GOTO 260
410 IF FX>153 THEN FX=FX-5
420 IF FX<153 THEN FX=FX+5
430 IF FY<135 THEN FY=FY+5
440 IF FY>135 THEN FY=FY-5
450 POKE V+4,FX:POKE V+5,FY:IF PEEK
   (V+30)=246 THEN 550
460 RETURN
470 IF S$="4" THENL1=118:L2=120
480 IF S$="2" THENL1=175:L2=120
490 IF S$="1" THENL1=145:L2=95
500 If S$="3" THENL1=145:L2=145
510 POKE V+6,L1:POKE V+7,L2:
   FU=FU-1:POKE V+21,255:
   IF FU<0THEN 540
520 IF PEEK(V+30)=252 THEN F=0:
   GOTO 240
530 GOTO 260
540 PRINT TAB(4)"▣▤YOU HAVE RUN
   OUT OF FUEL !":GOTO 560
550 PRINT TAB(11)"▣▤YOU'VE BEEN
   HIT !"
560 FOR ZZ=1TO10:FOR T=1TO100:
   NEXT:POKE V+21,247:FOR T=1TO100:
   NEXT:POKE V+21,0
570 NEXT:PRINT "♥"TAB(12)"▣▣▣▣
   ▣▣▣▣▣▣▣▣▨◪HIT
   SPACE BAR"
580 GET X$:IFX $<>"□" THEN 580
590 RUN 230
```

The first line of the program reserves some space for the sprite in the Commodore's memory, so that the BASIC program you use to move and operate the sprite cannot corrupt the sprite program itself.

Lines 2Ø to 14Ø contain the sprite information for the space station, shields, missiles and alien. Lines 21Ø and 22Ø set the sprite *pointers* and put the sprite DATA into memory. Each sprite occupies 64 bytes (3 times 21, plus one extra) and there are five of them—this explains the significance of these values in these two program lines.

The next two lines initialize the computer, setting the variables, the various sprite positions, auto-key repeat, and colours.

The program continues in Line 25Ø by creating a random position for the alien sprite, appearing along one line at the top or one line at the bottom depending on the value obtained by the RND function.

The program is seen to start in Line 26Ø with the screen display off, to begin with, the fuel remaining and the alien. This dispatches a missile and then moves off. Lines 36Ø–4ØØ activate the space station shields depending on which of the keys 1, 2, 3 and 4 is being pressed. If any of the shields is on, the condition in Line 39Ø is satisfied and the program jumps to the routine in Lines 47Ø–53Ø which controls the location of the relevant shield sprite.

All the time the missile is homing in on its target, in a routine spanning Lines 41Ø–46Ø. If the missile sprite succeeds in reaching the centre of the space station without interruption—in other words, if a sprite collision is not detected in Line 52Ø—the condition in Line 45Ø is satisfied. The game-end routine in Lines 54Ø onwards starts by displaying a message then flashes the screen display before offering you another go.

Each time the screen is activated, the fuel counter—variable FU—is decreased. The embedded cursor-left controls of the fuel display PRINT statement in Line 26Ø backspaces the cursor, effectively wiping out the previous fuel figure before adding the new value of FU each time the program returns to this point. When the value of FU is less than Ø, in Line 51Ø, a branch to the relevant part of the game-end routine is made.

ARRAYS-THE INFORMATION STORES

Race track statistics, names and dates—handling records like these is where the computer excels. And one of the most effective tools for processing them is the array

Arrays are the computer programmer's method of handling large amounts of closely-related information—for example, long lists of club members with their addresses and subscriptions, complex financial records, and even lists of characters, weapons or treasure for an adventure game.

To store individually each piece of information in such a long list, you could enter a massive number of LET statements. But this would create a long program and a lot of typing.

What the array does is to store the information in a far more compact form. Instead of having a different variable for each item of information, you use a common one—A, say. And to differentiate between each item, you merely use a figure (or sometimes letter) in brackets. So the first item is called A(1), the second A(2), the third A(3) and so on.

Not only does the array make your storage system more compact, but it also allows you to change any item—a new telephone number, for example—with an absolute minimum of trouble.

SETTING UP AN ARRAY

Before you can use an array, you have to tell the computer how big it will be, so that it can reserve enough space in its memory. This is done with the DIM (for 'dimension') statement, as in this line:

```
1Ø DIM A(3)
```

On the ZX81, use capital A.

```
1Ø DIM a(4)
```

This tells the computer that this particular array will have four elements, or variables. On Acorn, Commodore, Dragon and Tandy machines, they will be numbered from A(Ø) to A(3). On the Sinclairs which do not accept a(Ø), they will be numbered from a(1) to a(4).

The number of elements you use can be almost as large as you like—thousands, maybe. You need not actually use all the elements you reserve, so it is usually wise to overestimate. But bear in mind that the memory space thus reserved will no longer be available for other variables, so do not overdo it or you will get an 'out of memory' report.

ASSIGNING THE VALUES

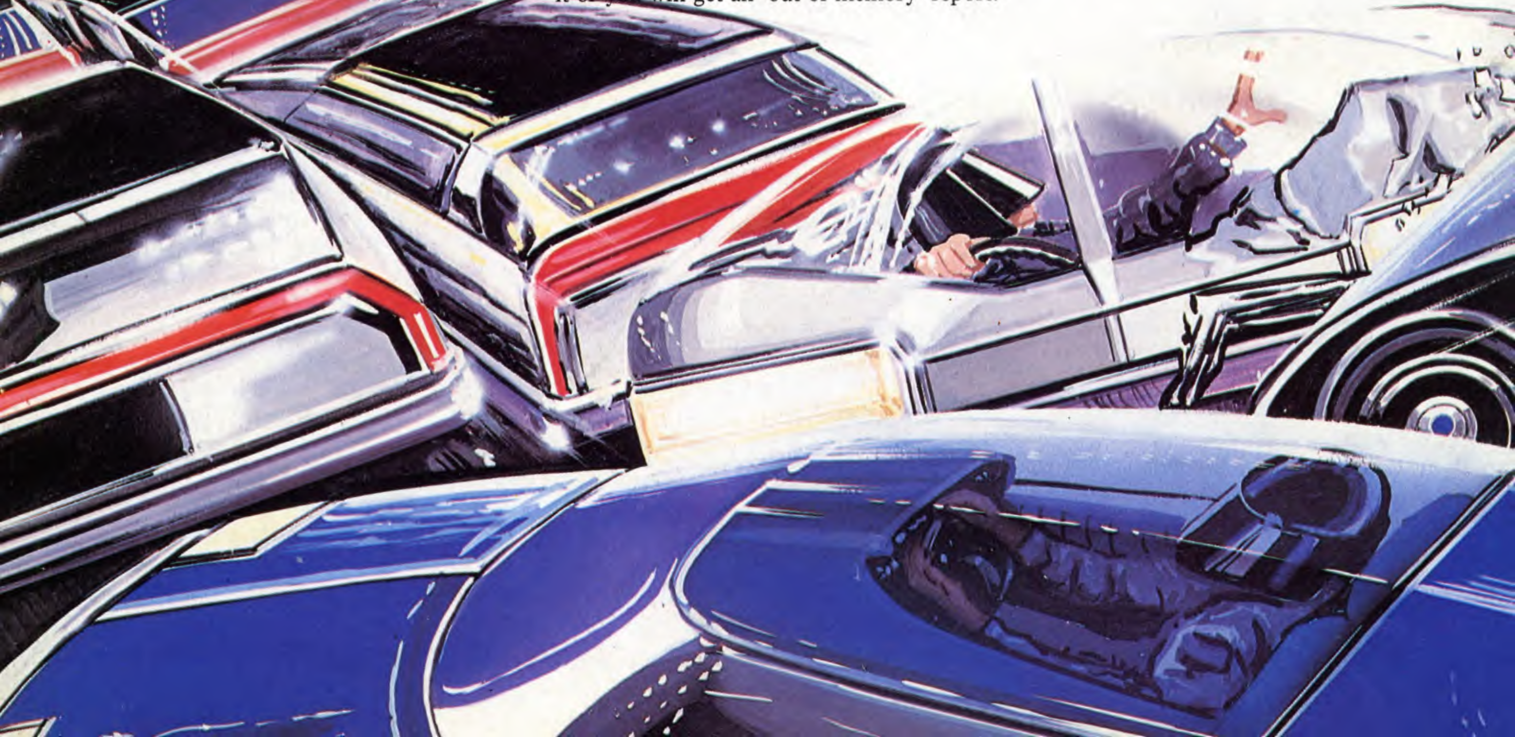
The next job is to assign values to each element. If, for example, the variables represented the screen positions at which text or a graphic were to be PRINTed, the next line might look like this:

```
2Ø LET A(Ø)=Ø: LET A(1)=2:
   LET A(2)=1Ø: LET A(3)=2Ø
```

```
2Ø LET a(1)=Ø: LET a(2)=2:
   LET a(3)=1Ø: LET a(4)=2Ø
```

```
2Ø LET A(1)=Ø
3Ø LET A(2)=2
4Ø LET A(3)=1Ø
5Ø LET A(4)=2Ø
```

So far, this represents no saving of time or memory space—in fact, the program is a bit longer than if you had used ordinary LET statements. But look what happens when you have values which are different each time the program is RUN:



- SETTING UP AND USING AN ARRAY
- THE DIM STATEMENT
- ASSIGNING VALUES
- HANDLING NAMES
- HOW TO USE DATA IN AN ARRAY
- ANALYSING THE INFORMATION
- THE USE OF ARRAYS IN GAMES PROGRAMS

```
1Ø DIM A(3)
2Ø PRINT "What are the values?"
3Ø INPUT A(Ø), A(1), A(2), A(3)
```

```
1Ø DIM a(4)
2Ø PRINT "What are the values?"
3Ø INPUT a(1),a(2),a(3),a(4)
```

```
1Ø DIM A(4)
2Ø PRINT "WHAT ARE THE VALUES"
3Ø INPUT A(1)
4Ø INPUT A(2)
5Ø INPUT A(3)
6Ø INPUT A(4)
```

Each time you RUN the program, the computer will ask what values each variable is to have this time. And all you need to do is to type a number (followed, of course, by ENTER or RETURN) each time it asks.

And when you have really large numbers of elements, the time saving is really worthwhile. Suppose, for example, you wanted to enter a hundred numbers. This simple program would be all you'd need:

```
1Ø DIM A(99)
2Ø FOR N=Ø TO 99
3Ø INPUT A(N)
4Ø NEXT N
```

On the ZX81, type entirely in capitals.

```
1Ø DIM a(1ØØ)
2Ø FOR n=1 TO 1ØØ
3Ø INPUT a(n)
4Ø NEXT n
```

HANDLING NAMES

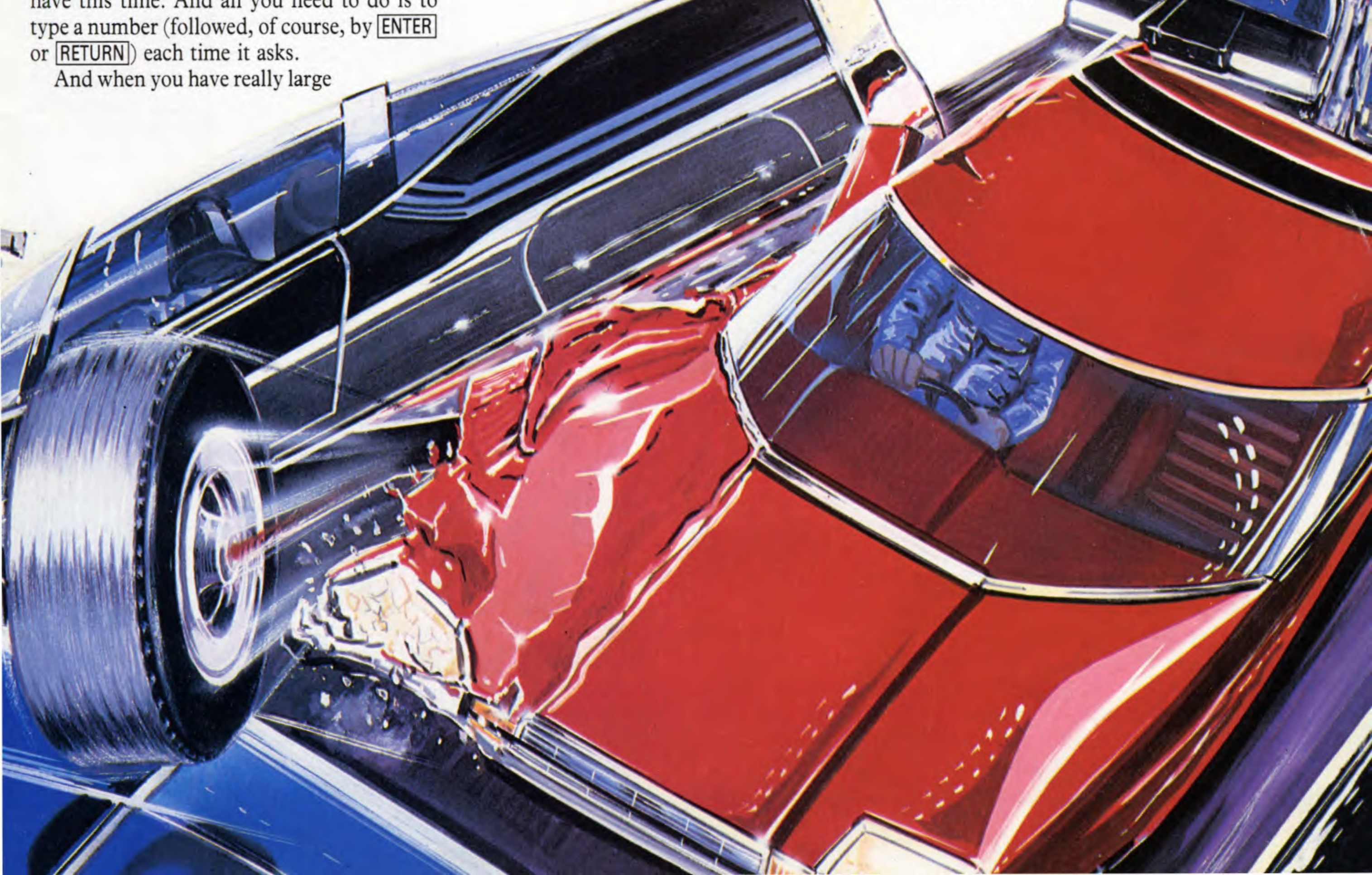
The type of array described so far lets you handle numbers only. If you want to handle names and numbers, you must set up two arrays: one for the names and the other for the numbers. The names array on three of our machines would look like this:

```
1Ø DIM A$(5)
```

On the Spectrum, it is also necessary to specify the maximum length that a name can be. So if your largest name were to have 1Ø characters, the names array would look like this:

On the ZX81, use capitals.

```
1Ø DIM a$ (6,1Ø)
```



In each case, the computer has reserved memory space for six names or six labels. To set up the INPUT loop, type:

```
2Ø FOR N=Ø TO 5
3Ø INPUT A$(N)
4Ø NEXT N
```

On the ZX81, type entirely in capitals.

```
2Ø FOR n=1 TO 6
3Ø INPUT a$(n)
4Ø NEXT n
```

RUN the program and enter each name or letter, followed by ENTER or RETURN. Then verify it by adding the following line and reRUNning:

```
35 PRINT A$(N)
```

On the ZX81, type this in capitals.

```
35 PRINT a$(n)
```

Even if the names are long, it is not difficult to type in six of them. And it is still quite quick, even if you have a hundred names.

Suppose, for instance, you were carrying out a survey of a racing circuit and you wished to store the names of the corners and the number of crashes that ocurred at each one during a racing season. You might easily need to enter a couple of dozen corners. And other examples may require even more entries than this. But the principle is the same as in the following program to enter just six. This method gets the computer to READ a store of DATA and so cannot be used on the ZX81, which does not have this facility:

```
1Ø DIM A$(5)
2Ø FOR N=Ø TO 5
3Ø READ A$(N)
4Ø NEXT N
5Ø DATA FIVE MILE, WELL PASS, BROOK
   HILL, PETERS ROAD, CROSSWAYS,
   ROWLANDS
```

```
1Ø DIM a$(6,11)
2Ø FOR n=1 TO 6
3Ø READ a$(n)
4Ø NEXT n
5Ø DATA "FIVE MILE", "WELL PASS",
   "BROOK HILL", "PETERS ROAD",
   "CROSSWAYS", "ROWLANDS"
```

The names are read from the DATA statements in Line 5Ø and stored into the array at Line 1Ø. So each time the program is RUN, the same names are entered automatically. If you had 1ØØ junctions, you would change Line 1Ø to DIM A$ (99) and Line 2Ø to FOR N=Ø TO 99 (for Acorn, Dragon, Tandy or Commodore) or DIM a$(1ØØ,11) and FOR n=1 TO 1ØØ (for Spectrum). Then you would add the other 94 junctions to the DATA statement.

Next, you could set up an array to hold the numbers of accidents:

```
6Ø DIM A(5)
7Ø FOR N=Ø TO 5
8Ø READ A(N)
9Ø NEXT N
1ØØ DATA Ø, 2, 5, 1, 3, 6
```

```
6Ø DIM a(6)
7Ø FOR n=1 TO 6
8Ø READ a(n)
9Ø NEXT n
1ØØ DATA Ø,2,5,1,3,6
```

If you like, you can ignore the zero element of an array on the Acorn, Commodore, Dragon and Tandy. So in the last example you can write DIM A$(6) and then number the items from 1 to 6. It means A$(Ø) is empty, but it is more natural to count from 1 rather than Ø.

USING ARRAYS

The computer now knows how many accidents occurred at each bend. But how can this information be manipulated?

The first thing you might want the computer to do is PRINT a list of all the bends and number of accidents—if only to check that you have keyed them in correctly. Type in the following:

```
21Ø FOR N=Ø TO 5
22Ø PRINT A$(N), A(N)
23Ø NEXT N
```

```
21Ø FOR n=1 TO 6
22Ø PRINT a$(n), a(n)
23Ø NEXT n
```

Lines 21Ø and 23Ø loop through the list, while line 22Ø PRINTs out the names and numbers stored in the arrays. If you find you've made a mistake then correct it now.

When you come to analyse the results of the survey, you will want to answer questions such as 'How many crashes were there in all?' and 'Which are the safest corners?'. The lines to find the total number of accidents might look like this:

```
3ØØ PRINT"♥"
31Ø LET TL=Ø
32Ø FOR N=Ø TO 5
33Ø LET TL=TL+A(N)
34Ø NEXT N
35Ø PRINT"TOTAL NUMBER OF
    ACCIDENTS□";TL
```

```
3ØØ CLS
31Ø LET total=Ø
32Ø FOR N=Ø TO 5
33Ø LET total=total+A(N)
34Ø NEXT N
35Ø PRINT "Total number of
    accidents:□";total
```

```
3ØØ CLS
31Ø LET TL=Ø
32Ø FOR N=Ø TO 5
33Ø LET TL=TL+A(N)
34Ø NEXT N
35Ø PRINT "TOTAL NUMBER OF
    ACCIDENTS□";TL
```

```
300 CLS
310 LET total=Ø
320 FOR n=1 TO 6
330 LET total=total+a(n)
340 NEXT n
350 PRINT "Total number of
    accidents:□";total
```

ANALYSING INFORMATION

Imagine how useful the few lines above would be if you had a road system with 100 or even 1000 bends, instead of just six.

And when you consider that the information in an array can not only be stored, but just as easily analysed, you'll see why the arrav is such a powerful tool—in everything from household budgeting (see pages 136 to 143) to international finance. For an example of such analysis, type in these extra lines.

```
400 FOR N= Ø TO 5
410 IF A(N)>3 THEN PRINT A$(N),A(N)
420 NEXT N
```

```
400 FOR n=1 TO 6
410 IF a(n)>3 THEN PRINT a$(n),a(n)
420 NEXT n
```

These lines PRINT a list of corners at which more than three accidents have occurred. In our example of six corners, these are BROOK HILL and ROWLANDS. If you change the 3 in Line 41Ø to 5 and RUN the program again, ROWLANDS is PRINTed. (Any number greater than 6, the largest value, would cause nothing to be PRINTed.)

The information stored in the arrays could just as easily be a list of families in a town, together with statistics such as number of children, income bracket and number of cars. Once these have been entered, they can be sorted into groups. You can even ask the computer to find a single name—even if you can remember only the initial letter. To show how this would work add these lines to the program:

```
600 FOR N=Ø TO 5
610 B$=A$(N)
620 IF LEFT$(B$,1)="P" THEN PRINT B$
630 NEXT N
```

```
600 FOR n=1 TO 6
620 IF a$(n,1)="P" THEN PRINT a$(n)
630 NEXT n
```

Lines 6ØØ and 63Ø set up the loop to look at each element of the array. As each name is read, Line 62Ø checks the first character and if it is a 'P' then the whole name is PRINTed.

In this case, it is PETERS ROAD that is PRINTed because it is the only junction beginning with 'P'. In a 'list of families' program, it could just as easily be all the Smiths, or everyone with more than one car. And with multi-dimensioned arrays, the subject of the next article on arrays, you can cross-index too—for example, you can find the entry for everyone whose name starts with 'A', who lives at street number 21 and whose dog is an alsatian!

ARRAYS FOR GAMES

Adventure games are one case where the array really comes into its own. Invariably, an adventure includes a number of locations, or scenes that the player visits. At each location, instructions are PRINTed on the screen to guide the player through the game. All these locations are related, so they are best stored in a *string* array—A$ followed by a number, for example A$(9).

The routes the player chooses to each location, such as 'north', fit well into a second string array; the objects, such as 'torch' and 'key', into a third; the verbs, such as 'take', 'kill' and 'dig', into a fourth.

At some locations, the player can collect objects, such as gold coins, which count towards the final score. These objects are stored in one *numeric* array— 'A' followed by a number, such as A(7)— and the number of objects the player is carrying in another.

In essence, then, a text-only adventure consists of a number of arrays which are manipulated by the program.

The development of adventure games needs a whole series of articles. This will be covered in Games Programming. But in the meantime, here is an example:

```
10 LET G=14
20 DIM A$(G),A(G)
30 FOR Z=1 TO G
40 READ A$(Z)
50 LET A(Z)=Z
60 NEXT Z
70 FOR X=G TO 2 STEP -1
80 LET Q=RND(X)
90 LET T=A(X):LET A(X)=A(Q):
   LET A(Q)=T
100 NEXT X
110 FOR T=1 TO G:PRINT "ROOM□";
   T;"□HAS A□";A$(A□(T)): NEXT T
120 DATA ROPE,SWORD,SPANNER,
   KNIFE,GUN,KEY,TORCH,CAR,
   WHIP,WAND,BOMB,BOOK,MODEL
   SHIP,ROBOT
```

```
10 LET G=14:PRINT CHR$(147)
20 DIM A$(G),A(G)
30 FOR Z=1 TO G
40 READ A$(Z)
50 LET A(Z)=Z
60 NEXT Z
70 FOR X=G TO 2 STEP-1
80 LET Q= INT( RND(1)*X)+1
90 LET T=A(X):LET A(X)=A(Q):LET A(Q)=T
100 NEXT X
110 FOR T=1 TO G:PRINT "ROOM";
   T;"HAS A□";A$(A(T)):NEXT T
120 DATA ROPE,SWORD,SPANNER,
   KNIFE,GUN,KEY,TORCH,CAR,
   WHIP,WAND,BOMB,BOOK
130 DATA MODEL SHIP,ROBOT
```

```
10 LET g=14
20 DIM a$(g,1Ø): DIM a(g)
30 FOR z=1 TO g
40 READ a$(z)
50 LET a(z)=z
60 NEXT z
70 FOR x=g TO 2 STEP -1
80 LET q=INT (RND*x)+1
90 LET t=a(x): LET a(x)=a(q):
   LET a(q)=t
100 NEXT x
110 FOR t=1 TO g: PRINT "Room□";
   t;"□has a□";a$(a(t)): NEXT t
120 DATA "rope","sword","spanner",
   "knife","gun","key","torch",
   "car","whip","wand","bomb",
   "book","model ship","robot"
```

Line 2Ø sets up a string array and a numeric array. Line 4Ø reads a list of objects, and Line 5Ø labels rooms. Lines 7Ø to 1ØØ assign an object randomly to each room, and Line 11Ø PRINTs the result.

HANDLING HEXA-DECIMAL ARITHMETIC

No sooner have you learnt to count on one finger than you have to learn to count on 16! But even if you don't have sixteen fingers you will find handling hex much easier than coping with chains of Øs and 1s

Though deep in their circuits computers do all their arithmetic in binary, using a number system composed entirely of Øs and 1s creates certain difficulties for human operators.

Reasonably-sized numbers soon end up with more noughts than a doughnut factory. And long series of Øs and 1s are not easy to key in. It is very easy to make a mistake and very difficult to spot one.

The way round this is for the operator to use a number system with yet another base.

Hexadecimal—or hex—numbers are numbers to the base 16. These are close enough to decimal, or ten-based, numbers to make them relatively easy for a human to handle.

Further, 16 is 2 × 2 × 2 × 2, which means that conversion between binary and hex is simple. Decimal 16 is 1Ø in hex and 1ØØØØ in binary. And every number from Ø to 15 is represented by a four-digit binary number.

To use a number system with a base bigger than ten you have to define new digits.

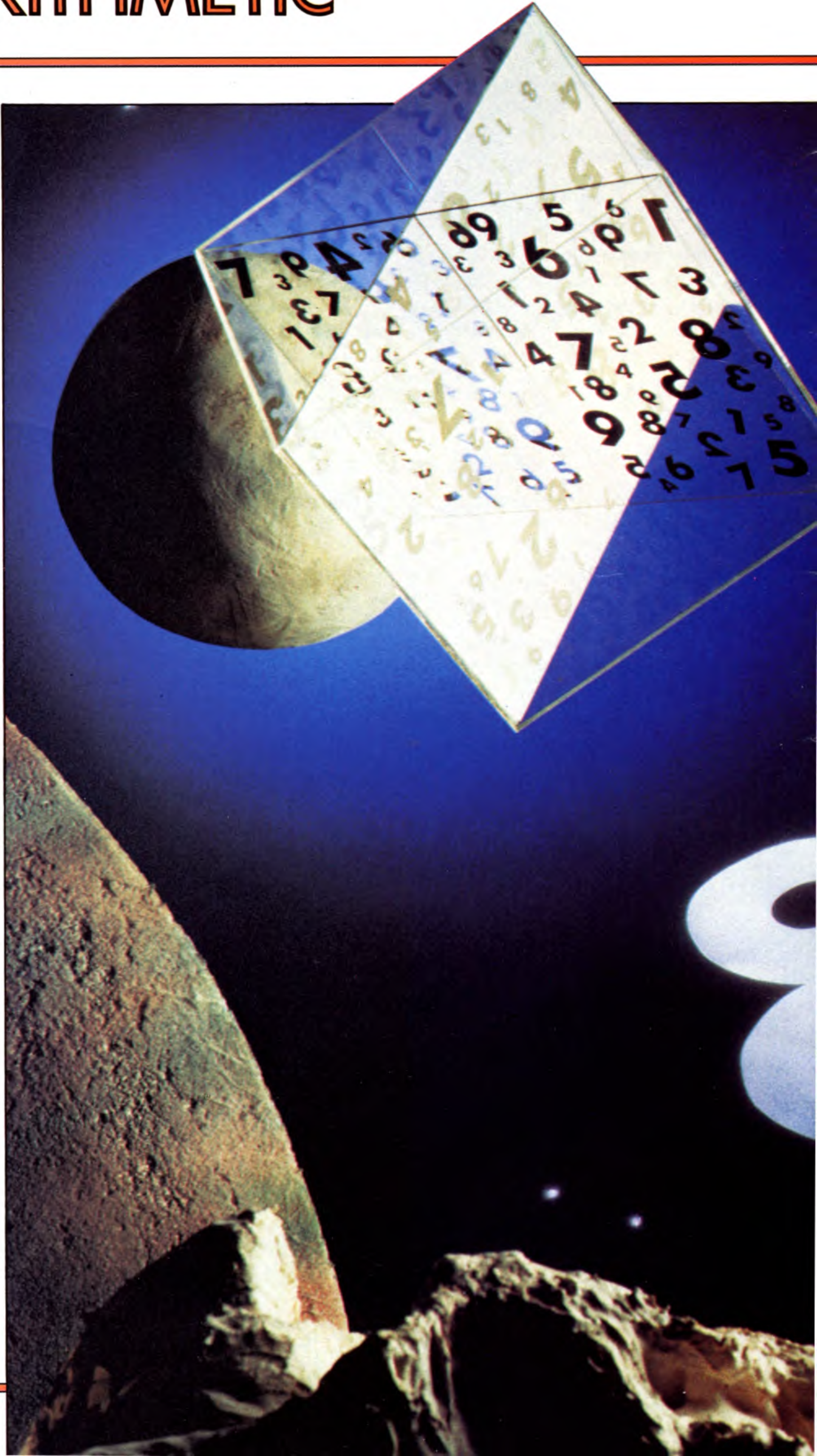
In hex, ten is represented by A, eleven by B, twelve by C, thirteen by D, fourteen by E and fifteen by F.

BINARY-HEX CONVERSION

To convert into hex the eight-bit binary numbers that home computers use is particularly easy. You break the number into two four-digit strings. Then, as explained on pages 38 and 39, the first four digits translate directly into one hex digit, and the last four into another hex digit.

Translating decimal into hex is more difficult. To do this you divide the decimal number successively by 16. The remainders after each division give the hex digits

For example, when you divide 1226 by 16 you get 76 with 1Ø left over. 1Ø is A in hex. 76 divided by 16 is 4 remainder 12. 12 is C in hex. And 4 divided by 16 is Ø remainder 4. So 1226 in decimal is 4CA in hex.



- WHY HEXADECIMAL IS USED
- COUNTING IN 16S
- THE RELATIONSHIP BETWEEN HEX AND BINARY
- EASY CONVERSION FROM BINARY TO HEX
- CONVERTING FROM DECIMAL TO HEX

The following program is quite long, but is worth keying in because it will help establish in your mind how this conversion works:

S

```
20 CLS
25 PLOT 140,0: DRAW 0,160
30 PRINT INVERSE 1;AT 0,8;"□BIN,
   DEC, HEX□"
40 PRINT INVERSE 1;AT 4,2;
   "□□BINARY:□□□□□"
50 PRINT INVERSE 1;AT 9,2;
   "□□DECIMAL:□□□□"
60 PRINT AT 10,5;"+□□□+
   □□□+□□□+□□□+
   □□□+□□□+"
70 PRINT INVERSE 1;AT 17,2;
   "□□HEXADECIMAL:"
80 PRINT AT 18,4;"+□□+
   □□+□□="
90 PRINT AT 18,20;"+□□+
   □□+□□="
100 LET no=0
110 GOTO 150
120 LET a$=INKEY$: IF a$="" THEN
   GOTO 120
130 IF a$="□" THEN LET no=no+1: IF
   no=256 THEN LET no=0
135 IF a$="b" THEN LET no=no-1: IF
   no=-1 THEN LET no=255
140 IF a$="b" OR a$="□" THEN GOTO
   150
145 INPUT "?";no
150 GOSUB 170: GOSUB 250
160 GOTO 120
170 LET nu=no: LET c=128
175 FOR x=0 TO 7
180 LET n=0: IF nu>=c THEN LET n=1:
   LET nu=nu-c
190 LET c=c/2
200 PRINT AT 5,2+4*x;n
210 IF n=1 THEN PRINT AT
   10,2+4*x;c*2
220 IF n=0 THEN PRINT AT
   10,2+4*x;"0□□"
230 NEXT x
235 PRINT AT 13,6;"DECIMAL
   TOTAL=□";no;"□□"
240 RETURN
250 LET hi=INT (no/16): LET hh=hi
260 LET lo=(no-hi*16): LET ll=lo:
   IF lo>9 THEN LET lo=lo+7
265 IF hi>9 THEN LET hi=hi+7
270 LET hi=hi+48: LET lo=lo+48
280 PRINT AT 18,14;CHR$ hi;AT
   18,30;CHR$ lo
290 LET c=8
300 FOR x=0 TO 3
310 LET n=0: IF hh>=c THEN LET
   n=c: LET hh=hh-c
315 LET m=0: IF ll>=c THEN LET
   m=c: LET ll=ll-c
320 LET c=c/2
330 PRINT AT 18,2+x*3;n;AT
   18,18+x*3;m
340 NEXT x
400 PRINT AT 21,6;"HEX TOTAL=□";
```

Microtip

Instant conversions

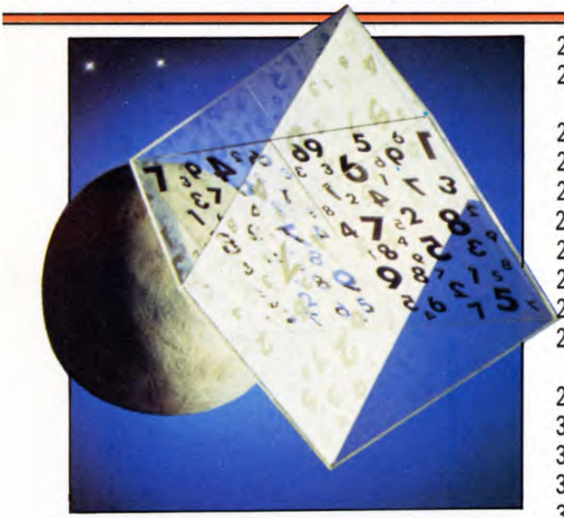
The BBC B, Acorn Electron, Dragon and Tandy computers have inbuilt programs to do decimal-to-hex conversions. To get a hex number, all you need to do is:

Type PRINT~, followed by the decimal number you want. Then press RETURN.

T

Type PRINT HEX$, followed in brackets by the decimal number you want converted—eg PRINT HEX$ (255)—then press RETURN.

If, on the other hand, you want to enter hex numbers as part of a program (when entering DATA statements, for example), these machines will accept them quite happily. On the Acorn machines, you must type & before the hex number; on the Dragon and Tandy, you must type &H before the number. The computers will then convert the hex into decimal for use during their subsequent calculations.



```
   CHR$ hi;CHR$ lo
500 RETURN
```

```
20 CLS
30 PRINT AT 0,9;"BIN, DEC, HEX"
40 PRINT AT 4,4;"BINARY:"
50 PRINT AT 9,4;"DECIMAL:"
60 PRINT AT 10,5;"+□□□+□□□
    +□□□+□□□+□□□
    +□□□
70 PRINT AT 17,4;"HEXADECIMAL:"
80 PRINT AT 18,4;"+□□+□□
    +□□="
90 PRINT AT 18,20;"+□□+□□
    +□□="
100 LET NO=0
110 GOTO 150
120 LET A$=INKEY$
125 IF A$="" THEN GOTO 120
130 IF A$<>"F" THEN GOTO 135
131 LET NO=NO+1
132 IF NO=256 THEN LET NO=0
135 IF A$<>"B" THEN GOTO 140
136 LET NO=NO1
137 IF NO=-1 THEN LET NO=255
140 IF A$="B" OR A$="F" THEN
   GOTO 150
145 INPUT NO
150 GOSUB 170
155 GOSUB 250
160 GOTO 120
170 LET NU=NO
171 LET C=128
175 FOR X=0 TO 7
180 LET N=0
185 IF NU>=C THEN LET N=1
186 IF NU>=C THEN LET NU=NU-C
190 LET C=C/2
200 PRINT AT 5,2+4*X;N
210 IF N=1 THEN PRINT AT 10,2+4*X;
   C*2
220 IF N=0 THEN PRINT AT 10,2+4*X;
   "0□□"
230 NEXT X
235 PRINT AT 13,6;"DECIMAL TOTAL
   =□";NO;"□□"
240 RETURN
250 LET HI=INT(NO/16)
255 LET HH=HI
260 LET LO=(NO-(HI*16))
261 LET LL=LO
270 LET HI=HI+28
275 LET LO=LO+28
280 PRINT AT 18,14; CHR$ HI;
   AT 18,30; CHR$ LO
290 LET C=8
300 FOR X=0 TO 3
310 LET N=0
311 IF HH>=C THEN LET N=C
312 IF HH>=C THEN LET HH=HH-C
315 LET M=0
316 IF LL>=C THEN LET M=C
317 IF LL>=C THEN LET LL=LL-C
320 LET C=C/2
330 PRINT AT 18,2+X*3;N;AT 18,18+X*3;M
340 NEXT X
400 PRINT AT 21,6;"HEX TOTAL=□";
   CHR$ HI; CHR$ LO
410 GOSUB 145
500 RETURN
```

```
10 MODE6
20 VDU23;8202;0;0;0;
30 PRINTTAB(13,2)"BIN, DEC, HEX"
40 PRINTTAB(13,3)STRING$
   (13,CHR$(224))
50 PRINTTAB(5,12);"+□□□+□□□+
   □□□□□+□□□+□□□+
   □□□+□□□="TAB(5,17)"+
   □□□+□□□+□□=
   □□□□□□□+□□□+□□□
   +□□="
60 PRINTTAB(1,5)"Binary□:"TAB(1,10)
   "Decimal□:"TAB(1,15)
   "Hexadecimal□:"TAB(12,20)
   "Hex Number□="
70 ?&70=0
80 T=?&70:PROCBIN:PROCDEC:
   PROCHEX
90 *FX21,0
95 G=GET
100 IF G=32 THEN?&70=?&70+1:
   GOTO80
105 IF G=66 THEN?&70=?&70-1:
   GOTO80
110 PRINTTAB(0,23);:INPUT?&70:
   PRINTTAB(0,23)STRING$(39,"□");:
   GOTO80
120 DEF PROCBIN
130 FOR X=0 TO 7
140 IF -(T AND 2∧X) THEN PRINTTAB
   (34-X*4+(X>3)*2,7)"1"TAB(34-
   X*4+(X>3)*3+(X>6),12);(T AND
   2∧X) ELSE PRINTTAB(34-X*4+(X>3)
   *2,7) "0"TAB(34-X*4+(X>3)*2-
   2,12); "□□0"
150 NEXT X
160 ENDPROC
170 DEF PROCHEX
180 FOR X=4 TO 7
190 PRINT TAB(31-X*4,17);(T AND
   2∧X)/16
200 NEXT
210 FOR X=0 TO 3
220 PRINTTAB(34-X*4,17);(T AND 2∧X)
230 NEXT
240 X=(T AND 240)/16
250 A$=CHR$(X+48-7*(X>9))
260 PRINTTAB(18,17);A$
270 X=(T AND 15)
280 B$=CHR$(X+48-7*(X>9))
290 PRINTTAB(37,17);B$
300 PRINTTAB(26,20)A$+B$
310 ENDPROC
320 DEF PROCDEC
330 PRINTTAB(37,12);T"□□"
340 ENDPROC
```

```
20 PRINT "♥"CHR$(8):FOR Z=1 TO
   8:READ A(Z):NEXT Z:DATA 128,64,
   32,16,8,4,2,1
30 K$="0123456789ABCDEF":POKE 650,
   255:POKE 53280,0:POKE53281,0
40 PRINT "☰"TAB(13)"π BIN,DEC,HEX"
50 PRINT TAB(13)
   "▐ - - - - - - - - - - -"
```

Q+A

How do I convert from hex back into decimal?

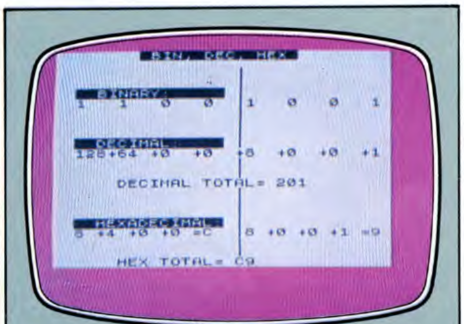
Each successive digit of a hex number is worth 16 times the digit to its right. So to convert a hex number like F6DA into decimal, you take the righthand digit and convert it into decimal notation. A is 10. The next digit to the left is worth 16 times more, so it must be converted into decimal notation and multiplied by 16. D is 13. 13 × 16 = 208. The next digit to the left is worth 16 times more again. 6 × 16 × 16 = 1536. And the last digit in this case must be multiplied by yet another 16. F is 15. 15 × 16 × 16 × 16 = 61440. So F6DA in hex is 10 + 208 + 1536 + 61440 or 63194 in decimal. Otherwise use the program here to convert the hex two digits at a time. Then multiply the left-hand pair by 256.

```
60 PRINT "▣▣▣πBINARY◣:▣"
70 PRINT "▣▣▣πDECIMAL◣:▣"
80 PRINT "π▣▣▣HEXADECIMAL
   ◣:▣"
90 PRINT"⊟◣▣▣▣▣▣▣":
   V$="": FOR Z=1 TO 8:V$=V$+LEFT$
   ("□"+ STR$(B(Z))+"□□",4):NEXT Z
100 PRINT MID$(V$,3,14)"□□"
   RIGHT$(V$,16)
110 PRINT "▣▣▣": V$="":FOR Z=1
   TO 8
120 V$=V$+LEFT$("+"+RIGHT$
   (STR$(D(Z)),LEN(STR$(D(Z)))-1)+
   "□□□",4):NEXT Z
130 PRINT MID$(V$,2,15);"□□□";
   MID$(V$,18,16)"=□□□□▌▌▌▌▌▌
   ▌▌";
140 PRINTD(1)+D(2)+D(3)+D(4)+
   D(5)+D(6)+D(7)+D(8)
150 PRINT "▣▣▣":V$="":FOR Z=1
   TO 8
160 V$=V$+LEFT$("+"+RIGHT$(STR$
   (H(Z)),LEN(STR$(H(Z)))-1)+
   "□□□",4):NEXT Z
170 X1$=MID$(K$,H(1)+H(2)+H(3)+
   H(4)+1,1):X2$=MID$(K$,H(5)+H(6)+
   H(7)+H(8)+1,1)
180 PRINT MID$(V$,2,15);"▌▌=
   ⊞"X1$"◣□□";MID$(V$,18,16);
   "=□⊞"X2$
190 PRINT,"▣▣▣☰□□HEX□
   TOTAL□:□□□□□□
   ▌▌▌▌▌▌▌▌▌▌⊞"
   ; X1$+X2$
200 GET A$:IF A$="" THEN 200
210 POKE 198,0:IF A$="□" THEN
   A=A+1:IF A>255 THEN A=0
220 IF A$="B" THEN A=A-1:IF A<0
   THEN A=255
230 IF A$<>"□" AND A$<>"B"
   THEN 350
240 AA=A
250 FOR Z=1TO 8:IF A(Z)<=AA THEN
   B(Z)=1:D(Z)=A(Z):AA=AA-A(Z):
   GOTO 270
260 B(Z)=0:D(Z)=0
270 NEXT
280 FORZ=1TO4:IFB(Z)=1THENH(Z)=
   A(Z+4):GOTO300
290 H(Z)=0
300 NEXT Z
310 FOR Z=1 TO 4:IF B(Z+4)=1 THEN
   H(Z+4)=A(Z+4):GOTO 330
320 H(Z+4)=0
330 NEXT Z
340 GOTO 90
350 I$="":PRINT "☰⊟▣▣▣▣▣
   ▣▣▣▣▣▣▣▣▣▣▣▣
   ▣▣▣▣▣□□□□□□□
   INPUT NUMBER? (0-255):
   □□□□▌▌▌▌▌▌";
```

```
360 FOR Z=1 TO 3
370 GET J$:PRINT "*▌▌□▌▌";:IF
   J$="" THEN 370
380 IF J$=CHR$(13) THEN 440
390 IF J$=CHR$(20) THEN 350
400 IF ASC(J$)<48 OR ASC(J$)>57
   THEN 370
410 I$=I$+J$:PRINT J$;:NEXT Z
420 GET J$:IF J$=CHR$(20) THEN 350
430 IF J$<>CHR$(13) THEN 420
440 IF VAL(I$)<0 OR VAL(I$)>255
   THEN 350
450 PRINT:PRINT "○□□□□□□□
   □□□□□□□□□□□□□
   □□□□□□□□□□□□□
   □□□□□□□□□";
460 A=VAL(I$):GOTO 240
```

```
10 PRINT"♡"CHR$(8)
20 FOR Z=1 TO 8:READ A(Z):NEXT:DATA
   128,64,32,16,8,4,2,1
30 K$="0123456789ABCDEF":POKE 650,
   255:POKE 36879,8
40 PRINT "⊟□□□□□π
   BIN,DEC,HEX"
50 PRINT "□□□□□☰-----
   ------"
60 PRINT "▣▣πBINARY◣:"
70 PRINT "▣▣▣πDECIMAL◣:"
80 PRINT "▣▣▣πHEXADECIMAL◣:"
90 PRINT "⊟▣▣▣▣▣◣":V$=
   "":FOR Z=1 TO 8:V$=V$+STR$
   (B(Z)):NEXTZ
100 PRINT LEFT$(V$,8)"□□□"
   RIGHT$(V$,8)
110 PRINT "▣▣":V$="":FORZ=1TO8
120 V$=V$+LEFT$("+"+RIGHT$
   (STR$(D(Z)),LEN(STR$(D(Z)))-1)
   +"□□□",4):NEXT Z
130 PRINT MID$(V$,1,15):PRINTMID$
   (V$,17,16)"▌▌=□□□□▌▌▌▌▌▌▌▌";
140 PRINT D(1)+D(2)+D(3)+D(4)+D(5)
   +D(6)+D(7)+D(8)
150 PRINT "▣":V$="":FORZ=1TO8
160 V$=V$+"+"+RIGHT$(STR$
   (H(Z)),1):NEXT Z
170 X1$=MID$(K$,H(1)+H(2)+H(3)
   +H(4)+1,1):X2$=MID$(K$,H(5)
   +H(6)+H(7)+H(8)+1,1)
180 PRINT LEFT$(V$,8):"=⊞"X1$;
   "□◣"RIGHT$(V$,8)"=⊞"X2$
190 PRINT"▣☰HEX TOTAL :□□□□
   □□▌▌▌▌▌▌▌▌▌▌⊞";
   X1$+X2$
200 GETA$:IFA$=""THEN200
210 POKE 198,0:IFA$="□"THENA=A+1:
   IFA>255THENA=0
220 IFA$="B"THENA=A-1:IFA<0THEN
   A=255
230 IF A$<>"□"ANDA$<>"B"THEN350
240 AA=A
250 FORZ=1TO8:IFA(Z)<=AATHENB(Z)
   =1:(DZ)=A(Z):AA=AA-A(Z):GOTO270
260 B(Z)=0:D(Z)=0
270 NEXT
```

```
280 FORZ=1TO4:IFB(Z)=1THENH(Z)=A
    (Z+4):GOTO300
290 H(Z)=Ø
300 NEXTZ
310 FORZ=1TO4:IFB(Z+4)=1THENH
    (Z+4)=A(Z+4):GOTO330
320 H(Z+4)=Ø
330 NEXTZ
340 GOTO9Ø
350 I$="":PRINT"■■■■
    ■■■■■■■■■■■
    ■■■■■■INPUT NUMBER?■
    Ø−255) ▓ > □□□□▮▮▮▮▮▮";
360 FORZ=1TO3
370 GETJ$:PRINT"*▮▮□▮▮";:IF J$=""
    THEN37Ø
380 IFJ$=CHR$(13)THEN44Ø
390 IFJ$=CHR$(2Ø)THEN35Ø
400 IFASC(J$)<48 ORASC(J$)>57
    THEN37Ø
410 I$=I$+J$:PRINTJ$;:
    NEXTZ
420 GETJ$:IFJ$=CHR$(29)THEN35Ø
430 IFJ$< >CHR$(13)THEN42Ø
440 IFVAL(I$)<ØORVAL(I$)>255
    THEN35Ø
450 PRINT:PRINT"●●□□□
    □□□□□□□□□□□
    □□□□□□□□□□□
    □□□";
460 A=VAL(I$):GOTO24Ø
```



```
        BIN, DEC, HEX
BINARY
1    1    Ø    Ø    1    Ø    Ø    1
DECIMAL
128+64 +Ø  +Ø  +8  +Ø  +Ø  +1
     DECIMAL TOTAL= 2Ø1
HEXADECIMAL
8 +4 +Ø +Ø =C   8 +Ø +Ø +1 =9
     HEX TOTAL= C9
```

This is how the Bin, Dec, Hex program looks on the Spectrum. The screen layout is not too different on the other machines. It is now easy to see how the three number systems work. When you RUN the program all three lines are set to zero. Press B, for back, and the Binary line will fill up with 1s. Underneath, the decimal line will fill up with the powers of 2. From right to left, you get 1—which is 2^0; 2—which is 2^1; 4—which is 2^2; and so on. The hex line works in the same way, only the two hex digits are computed independently.

```
20 CLSØ
30 PRINT@11, "BIN,DEC,HEX";
40 PRINT@68,"BINARY";
50 PRINT@196,"DECIMAL";
60 PRINT@ 323,"HEXADECIMAL";
70 PRINT@355,"+□□+□□+
    □□□=□";
80 PRINT@371,"+□□+□□+
    □□□=□";
90 FORJ=1TO15:POKE1Ø4Ø+32*J,
    175:NEXT
100 PRINT@45Ø, "HEX NUMBER=
    □□□□□";
110 PRINT@227,"+□□□+□□□+
    □□□□+□□□+□□□+
    □□□+"
120 GOTO17Ø
130 IN$=INKEY$:IFIN$="" THEN13Ø
140 IFIN$="□"THENNO=NO+1:
    NO=NO AND 255:GOTO17Ø
150 IFIN$="B" THENNO=NO−1:NO=NO
    AND 255:GOTO17Ø
160 GOSUB37Ø
170 GOSUB19Ø:GOSUB27Ø
180 GOTO13Ø
190 FORX=7TOØ STEP−1
200 IF(NO AND 2 ↑ X) THENN=1
    ELSEN=Ø
210 PRINT@125−X*4,N;
220 IFN=1 THENN=INT(2 ↑ X):
    N$=STR$(N):N$=MID$(N$,2,
    LEN(N$)−1) ELSEN$=RIGHT$
    ("□□Ø", LEN(STR$(2 ↑ X))−1)
230 PRINT@255−X*4−LEN(N$),N$;
240 NEXT
250 PRINT@279,"□=□";MID$
    (STR$(NO)+"□□",2,3);
260 RETURN
270 FORX=7 TO 4 STEP−1
280 PRINT@374−X*3,STR$((NO
    AND 2 ↑ X)/16);
290 NEXT
300 PRINT@367,HEX$(NO/16);
310 FORX=3TOØ STEP−1
320 PRINT@378−X*3,STR$(NO AND 2 ↑ X);
330 NEXT
340 PRINT@383,HEX$(NO AND 15);
350 POKE1488,PEEK(1391):POKE
    1489,PEEK(14Ø7)
360 RETURN
370 NU$="":PRINT@439,"?";
380 IN$=INKEY$:IF(IN$<"Ø"ORIN$>
    "9") ANDIN$< >CHR$(13)THEN
    GOTO 38Ø
390 IFIN$=CHR$(13) THENNO=VAL
    (NU$):IFNO>255 THEN37Ø
    ELSEPRINT@439,STRING$(5,
    CHR$(128));:RETURN
400 IFIN$< >CHR$(13) ANDLEN
    (NU$)>2 THEN38Ø
410 NU$=NU$+IN$:PRINT@441,
    MID$(NU$+"□□□",1,3);:GOTO38Ø
```

Once you've keyed in the program for your machine and RUN it, you will find that the binary, decimal and hex numbers are all set at zero. If you push the space bar a 1 will clock up in each base. Keep pushing and the computer will keep counting, adding 1 to each total at a time. Note that the decimal equivalent is computed by adding the value of each place that has a 1 in it in the binary.

The hex is computed by doing exactly the same thing, except that it takes four binary digits at a time.

Pressing the B key will subtract a 1 from each of the numbers and run the program backwards.

For the Spectrum and Commodores you can SAVE this program and use it to convert decimal numbers into hex numbers at any time. The quick way to do this conversion is to press any key on the keyboard except the space bar or B. A question mark will be displayed on the screen. Feed in any decimal number less than 255, press ENTER or RETURN, and the equivalent in binary and hex will be displayed.

You will note that the maximum number that can be represented by an eight-bit byte in binary is 11111111. This is 255 in decimal, and FF (the maximum two-digit number) in hex. Any number stored in a byte of your computer memory can be represented by a two-digit hex number. And machine code is made up entirely of these two-digit hex numbers.

LARGER NUMBERS

Your computer deals with numbers larger than 255 simply by breaking them in two parts and putting them into two adjacent memory locations. This will allow you to store any number up to FFFF in hex, or 65,535 in decimal. FFFF is an important number in home computers as it is the maximum number of addressable memory locations.

Larger numbers still can be stored by breaking the number into three or four hexadecimal bytes and storing them in succeeding memory locations. Which way round the bytes are stored is a matter of convention. The Sinclair, Commodore and Acorn computers store the lowest value byte in the lowest memory location and the highest in the highest. The Dragon and the Tandy store them the other way round.

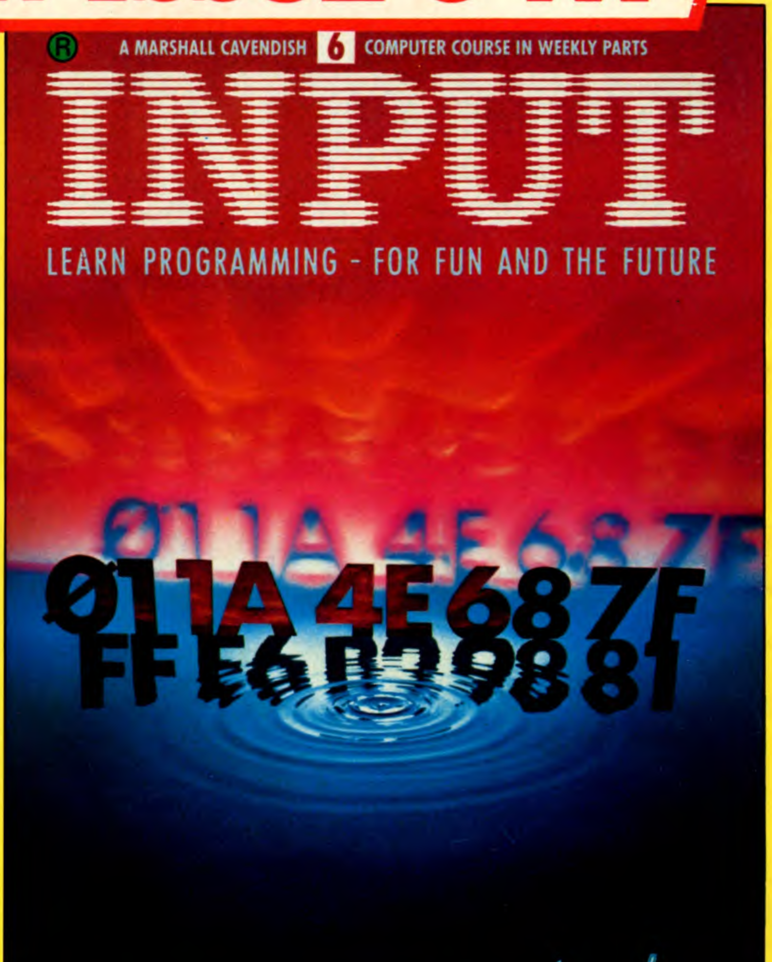
But how does hex represent negative numbers? That will be dealt with in the next part of this article.

CUMULATIVE INDEX

An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

The publishers accept no responsibility for unsolicited material sent for publication in INPUT. All tapes and written material should be accompanied by a stamped, self-addressed envelope.

COMING IN ISSUE 6 ...


A MARSHALL CAVENDISH 6 COMPUTER COURSE IN WEEKLY PARTS
INPUT
LEARN PROGRAMMING - FOR FUN AND THE FUTURE
Ø11A 4E687F

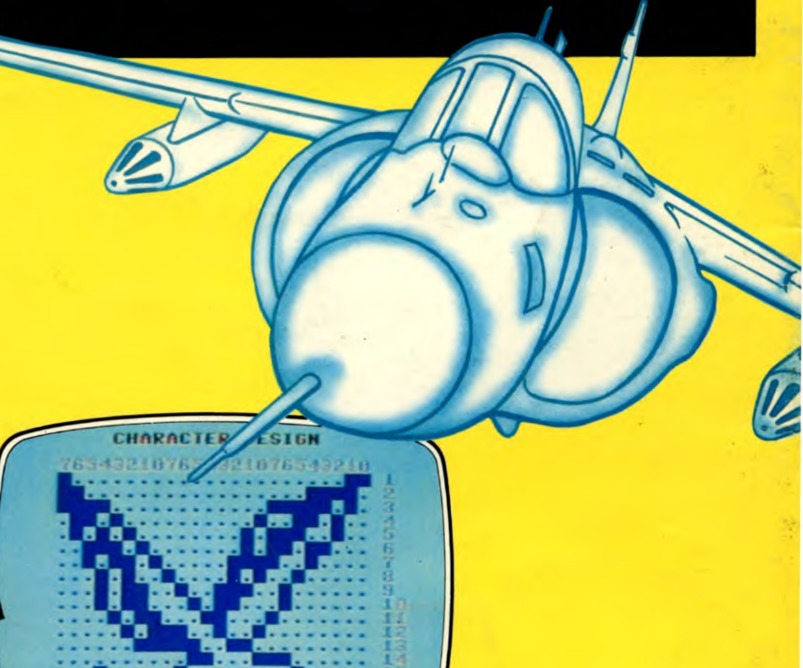
❑ *Improve your ability to draw* ***COMPUTER PICTURES*** *by learning about some of the more subtle uses for your computer's BASIC graphics commands*

❑ *Zap ... pow ... crash.* ***EXPLOSIONS*** *are a common feature of many arcade-type games. So here's how you can create convincing visual effects to add to your games—plus a couple of new routines*

❑ *Get one step nearer to understanding the mysteries of the numbers which make up machine code programs by looking at how the computer handles* ***NEGATIVE NUMBERS***

❑ *What do people mean when they talk about an 'elegant' or well-written program? Find out with your own guide to writing properly* ***STRUCTURED PROGRAMS***

❑ *Plus, for Commodore users, one of an occasional series on special features of individual machines. In this article, we look at the sprite graphics facility*



ASK YOUR NEWSAGENT FOR INPUT